段階的詳細化に基づく通信システムの仕様記述に関する研究

博士学位論文

論文題目　段階的詳細化に基づく
通信システムの仕様記述に
関する研究

提出者　東北大学大学院情報科学研究科
情報基礎科学 専攻

学籍番号　5ds6

氏名　細野 健太郎

指導教官	白鳥 則郎 教授
審査委員 （○印は主査）	○ 白鳥 則郎 教授 1 根元 義章 教授　2 牧野 正三 教授 3 ＿＿＿＿ 教授　4 ＿＿＿＿ 教授 5 ＿＿＿＿ 教授　6 ＿＿＿＿ 教授

(1)

平成７年度 博士学位論文

段階的詳細化に基づく通信システムの仕様記述に関する研究

東北大学大学院 情報科学研究科
情報基礎科学専攻
博士課程後期３年の課程

郷 健太郎

平成８年１月19日 提出

もくじ

第 1 章

序論

1.1 背景と目的

近年, 情報通信システム利用者の急激な増加にともない, システムに対する利用者要求が多種多様化してきている. これらの要求を満足させるため, 必然的にシステムが大規模・複雑化してきている. このような現状で高品質・高信頼なシステムを開発するために, その系統的なシステム設計方法論の確立が切望されている.

通信プロトコルや情報ネットワークのような大規模, 複雑なシステムの設計には, 一般に次の 2 つの方法がある.

1) 段階的詳細化による方法 (トップダウンアプローチ):
   これは, 構築しようとするシステムの要求仕様を, 段階的に分割, 詳細化を繰り返して行くことによって, 実システムを構築する方法である.

2) 部品合成による方法 (ボトムアップアプローチ):
   これは, システムの構造の単位をモジュールとし, 汎用的なモジュール群や特別なモジュール群を用意しておいて, これらを組合わせることで, 要求仕様に適合する実システムを構築する方法である.

本研究では, このうち 1) の段階的詳細化による方法を対象とする.

一方で, 高品質・高信頼なシステムを開発するためには, その設計工程への形式記述技法 (Formal Description Techniques: FDTs) [1, 2] の導入が有効である. この有効性は, FDTs の持つ次の 2 つの利点による. a) 自然言語や図表で記述された仕様が曖昧性を持つのに対し, FDTs によって記述された仕様は曖昧性を持たない. b)FDTs で記述された仕様に対しては, 実装する以前に仕様上での検証が可能になる.

本研究では FDTs の有効性を活かし, 段階的詳細化に基づく通信システムの仕様記述を研究対象とする. 特に, 段階的詳細化に基づく仕様記述を系統的に実現するために要求される基盤技術のうち, 次の 2 点に焦点をあてて研究を行う.

1) 系統的な仕様の分割法が必要である.

2) 開発工程の段階において, 前段階での仕様の変化を後段階へ効率的に伝える方法が必要である.

すなわち, 本研究での目的は次の 2 点である.

a) 仕様の等価性を保存する分割法を構成する.

b) 前段階での仕様の変化に対する後段階の仕様の自動変更法を構成する.

これらの目的を達成するために, 本論文では以下の項目を提案する.

まず a) のために, ISO で標準化された FDTs の一つである LOTOS[3, 4, 5] で記述された仕様の分割法を提案する. そして, この分割法が分割の前後で仕様の等価性を保存することを証明する. 次に, ここで提案した分割法を任意のプロセス代数仕様の分割法に一般化するために, ラベル付き遷移システムの分割法を構成する.

次に b) のために, ソフトウェアのやわらかい開発法という枠組を構成する. これは, 利用者要求や, システムが提供するサービスに変化が生じ

た場合，これらの変化を開発支援システムが自動的に吸収し，新しい利用者要求や，システムの新しいサービスを効率的に実現するための概念的枠組である．その具体的応用として，通信システムのサービス仕様の変化に対するプロトコル仕様の自動変更法を提案する．この手法によって変更されたプロトコル仕様は，サービス仕様の正当性を保存する．そのため本手法は，高信頼システムの設計工程における要求仕様の変更時に，特に有効となる．

1.2 論文の構成

本論文は，全 6 章と付録から構成されている．

第 1 章は序論であり，本研究の背景と目的について述べた．

第 2 章では，本論文で対象としている FDT について要約する．

第 3 章以下が本研究で得られた結果である． 第 3 章では LOTOS 仕様の分割問題を定義し，その問題を解決する分割アルゴリズムについて述べ，具体的な仕様の分割例を示す．

第 4 章では，第 3 章で得られた結果を一般化するために，LTS 仕様の分割法を構成する．

第 5 章では，段階的詳細化に適したやわらかい開発法について述べ，その具体的応用として，通信システムのサービス仕様の変化に対するプロトコル仕様の自動変更法を提案する．

第 6 章は本論文の結論である．

付録として，第 A 章で LOTOS の記述スタイルについて要約する．

第 2 章

形式記述技法

2.1 はじめに

本章では, 本研究の基盤となる形式記述技法について述べる. まず 2.2 節で形式記述技法の背景について述べ, 2.3 節で形式記述言語 LOTOS について述べる. 2.4 節では, LOTOS の意味を表現するために使われるラベル付き遷移システムについて述べる. 2.5 節では形式記述言語によって記述された仕様間の等価性について述べる. 2.6 節は本章のまとめである.

2.2 形式記述技法の背景

計算機科学において, 形式記述技法 (Formal Description Techniques: FDTs) が重要視されるようになった背景には, 情報通信システムの急激な発達がある. 多数のコンピュータを相互に接続し, 大規模なコンピュータネットワークを構成したいという要求の増加に対し, コンピュータ間の通信プロトコル (Protocol) を国際標準として制定する必要性がでてきた. この通信プロトコルの標準化を目的に, 国際標準化機構 (ISO) では, 開放型システム間相互接続 (Open Systems Interconnection: OSI) のプロジェクトを開始した. このプロジェクトの成果である通信プロトコルは, 従来自然言語と図表を用いて記述されていた. しかし, これら

が持つ曖昧さのために, 次の 2 つの問題が発生した.

1) 標準文書に規定されている同じプロトコルに対し, 複数のシステム設計者が異なった解釈のもとに構築し, 結果としてシステム間で相互に通信できない場合が生じた.

2) 通信プロトコルが複雑になるにつれ, その中にあるエラーを人手によって検証するのが不可能になった.

これらの問題を解決するために, FDTs に対する研究が活発に行われるようになった. すなわち, 曖昧性がなく機械的な処理が可能な形式的な言語と, これを用いた系統的なシステム仕様化法の研究がさかんになった.

FDTs は, 形式記述言語 (Formal Specification Language) と形式技法 (Formal Method) から構成される. これらのうち, 現在までに国際標準となっているものは前者の形式記述言語である. ISO では, プロセス代数に基づく LOTOS[3, 4, 5] と, 拡張状態遷移モデルに基づく ESTELLE[6] を標準化し, CCITT[1] では, SDL[7] を制定・勧告した. 後者の形式技法については, 現在研究過程にあり標準化までには至っていない.

FDTs は, 自然言語と図表を用いて仕様記述した際に問題となる点を解消する目的で開発されているため, FDTs をシステム設計工程に導入することで, 次のような利点が得られる.

1) 曖昧性のない, 厳密なシステム仕様が得られる.

2) システム中にあるエラーの機械的検証が可能になる.

一方で, FDTs は数学的厳密さを持つために, 記述性・理解性に乏しいという欠点がある. 標準化された 3 つの FDTs については, いずれも

[1]現在の名称は ITU-T である.

図的表現を備えているが, 個々のFDTに特有のモデルに対する考え方を与えるものではない. 従って, FDTsを設計工程に導入するためには, 個々のFDTに特有のモデルに対する考え方について, 設計者を教育する必要がある.

FDTsが持つ記述性・理解性の乏しさを解決する研究も活発に行われている. 最も多いアプローチはユーザフレンドリな仕様記述支援環境の構築である [11]. これに加えてLOTOSでは, 設計工程の段階に応じた仕様の記述スタイル (Specification Styles) [12] という概念を導入している. LOTOSの記述スタイルについては, 付録 A で説明する.

2.3 LOTOS

LOTOSは次の考え方に基づいて言語設計が行なわれている, すなわち「システムは, そのシステムとやりとりする外部から観測可能なイベント間の時間順序を規定することにより仕様化可能である」

LOTOSは以下の2つ要素から構成されている:

1) プロセスの振舞い (動作) に関する記述, ここで, プロセスとは, その環境 (対象としているプロセスの外側全体) における他のプロセスと通信を行なう抽象的な実体を指す.

2) データ構造とデータ値に関する記述.

1) はプロセスの動作を代数としてとらえるCCS (Calculus of Communicating System)[9] のモデルを基礎としている.

2) は抽象データ型 [10] の記述言語 ACT ONE を基礎としている.

プロセス間の通信はゲートと呼ばれる作用点で起こり, 通常, そこでデータ値の交換が行なわれる. 通信による相互作用の基本単位はアクション (あるいはゲート) と呼ばれ, ゲートとデータ値の組から構成される. アクションに対しては以下の2つ性質が仮定される:

a) アトミック性: これは (i) 瞬時発生的であり, (ii) それ以上細分化できず, (iii) 他のアクションとの重なりがないことを意味する.

b) 同期的: これは, 環境も含めて, 通信する 2 つ以上のプロセスが, 同時にその発生に関与すると言うことを意味する.

ゲートにおいてデータ値を交換しない通信の場合, ゲートのみがアクションを構成し, 単に同期を表現することになる.

以上の概念よると, プロセスの静的側面は図 2.1 のようなブラックボックスとして表現することができる. 同図において, a, b, c はプロセス P が環境と通信するゲートを表している.

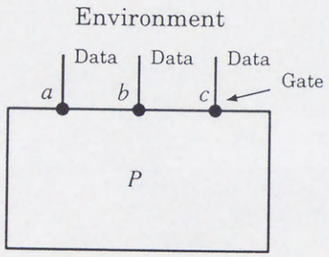


図 2.1: ブラックボックスとしてのプロセス

図 2.1 の例におけるプロセス P の動作は, 3 つのゲート a, b, c で起こる観測可能なアクションの系列を動作式として記述することによって定義される. アクションの系列は, 枝にアクションのラベルを付けた木を用いて表現できる (図 2.2 参照).

LOTOS は (1) プロセス間の同期のみを取り扱う Basic LOTOS [4] と, (2) データ値の交換を取り扱う Full LOTOS とに分けられるが, 本節では (1) の Basic LOTOS について説明する. (2) の Full LOTOS の詳細については, 文献 [3] を参照されたい.

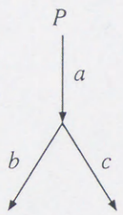


図 2.2: アクション木

2.3.1 構文

基本プロセス

どんなアクションも生起しない非能動的なプロセスは, 動作式

stop

によって表現される. これは, プロセスの動作が停止し, それ以上何もしないことを表わしている.

基本演算子

1) アクションプレフィックス $a;B$

プロセスがアクション a を生起することができ, 次に B によって表現される動作を行うことができる.

2) チョイス $B_1 \,[]\, B_2$

プロセス B_1 あるいは B_2 のいずれかのように動作する. どちらが選択されるかは, プロセスと環境との交信によって決まる.

プロセスインスタンシエーション

$$\mathrm{p}[g_1, \cdots, g_n]$$

は, プロセス名 p と実ゲート引数リスト $g_1, ..., g_n$ から成り, 自分自身あるいは他で定義されたプロセス p をインスタンシエートする. インスタンシエートされたプロセスの動作は, その対応するプロセス定義によって与えられる. インスタンシエーションの際の実ゲートに対応する仮ゲートを巻き込むアクションは, その実ゲートを巻き込むアクションの発生へと置き換えて解釈される. これはリラベリング (relabelling) と呼ばれる.

プロセスの並列合成

1) 同期しないプロセスの並列合成 $B_1 \,|||\, B_2$

これはプロセス B_1 と B_2 との独立的な並列合成を表わす. プロセス B_1 でアクションが起り得る時, あるいは, B_2 で起り得る時, $B_1 \,|||\, B_2$ でそのアクションが生起し得る. 一方, B_1 と B_2 の両方によって拒否されるどんなアクションも生起不能である.

2) 完全同期するプロセスの並列合成 $B_1 \,||\, B_2$

これはプロセス B_1 と B_2 との同期的な並列合成を表わす. プロセス B_1 と B_2 はすべてのアクションに関して同期しなければならない. B_1 と B_2 の両方で同じアクションが生起可能な時に限り, $B_1 \,||\, B_2$ でそのアクションが生起し得る. B_1 あるいは B_2 によって拒否されるどんなアクションも生起不能である.

3) 一般並列合成 $B_1|[g_1, ..., g_n]|B_2$

これは B_1 と B_2 とがゲート $g_1, ..., g_n$ を巻き込むアクションに関して同期しなければならないような一般的な並列合成を表す. この場合, 同期すべきアクションはそれらが関与するゲートのリストとして陽に与えらる.

2.3.2 意味

LOTOS の構文の意味は, 次節で述べるラベル付き遷移システムによって定義される. 前節で与えた LOTOS の各構文の意味は, 表 2.1 に示される.

2.4 ラベル付き遷移システム

LOTOS の動的な意味は, ラベル付き遷移の集まりであるラベル付き遷移システム (Labelled Transition System: LTS) [3, 8, 9] の観点で与えられる. LTS はグラフ構造を成し, それを開く (unfold) ことにより, アクション木との対応が得られる.

ラベル付き遷移は次のように表現される:

$$B \xrightarrow{a} B'$$

ここで, B と B' は動作式であり, a はアクションである. この遷移は「プロセスの動作式 B はアクション a を実行し B' になる」あるいは「B はアクション a の生起後 B' になる」ことを意味している.

LTS の形式的な定義は次のとおりである.

定義 1 (Labelled Transition System: LTS) LTS は 4 項組

$$(S, L, T, s_0)$$

である. ここで, S は空でない状態の可算集合; $L = L_v \cup \{\mathbf{i}\}$ であり, L_v は観測可能なアクションの可算集合, $\mathbf{i}$ は観測不可能な内部アクションである;

$$T = \{\xrightarrow{a} \mid \xrightarrow{a} \subseteq S \times S, a \in L\}$$

は遷移関係; $s_0 \in S$ は初期状態である.

ここで, 状態集合とアクション集合が有限でないことに注意する.

表 2.1: LOTOS の基本構文とその意味

(アクションプレフィックス)

$$a; B \xrightarrow{a} B$$

(チョイス)

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1 \,[]\, B_2 \xrightarrow{a} B_1'} \quad \frac{B_2 \xrightarrow{a} B_2'}{B_1 \,[]\, B_2 \xrightarrow{a} B_2'}$$

(プロセスインスタンシエーション)

$$\frac{Bp[g_1/g_1', ..., g_n/g_n'] \xrightarrow{a} B'}{p[g_1/g_1', ..., g_n/g_n'] \xrightarrow{a} B'}$$

ただし, "**process** $p[g_1', ..., g_n'] := Bp$ **endproc**" をプロセス p の定義とする.

(リラベリング)

$$\frac{B \xrightarrow{a} B'}{B[g_1/g_1', ..., g_n/g_n'] \xrightarrow{a'} B[g_1/g_1', ..., g_n/g_n']}$$

ただし, $a \notin \{g_1, ..., g_n, \delta\}$ の時 $a' = a$, $a = g_1'(1 \leq i \leq n)$ の時 $a' = g_i$.

(並列合成)

$$\frac{B_1 \xrightarrow{a} B_1', a \notin \{g_1, ..., g_n, \delta\}}{B_1 |[g_1, ..., g_n]| B_2 \xrightarrow{a} B_1' |[g_1, ..., g_n]| B_2}$$

$$\frac{B_2 \xrightarrow{a} B_2', a \notin \{g_1, ..., g_n, \delta\}}{B_1 |[g_1, ..., g_n]| B_2 \xrightarrow{a} B_1 |[g_1, ..., g_n]| B_2'}$$

$$\frac{B_1 \xrightarrow{a} B_1', B_2 \xrightarrow{a} B_2', a \in \{g_1, ..., g_n, \delta\}}{B_1 |[g_1, ..., g_n]| B_2 \xrightarrow{a} B_1' |[g_1, ..., g_n]| B_2'}$$

$$\frac{B_1 \;|[]|\; B_2 \xrightarrow{a} B_1'}{B_1 \;|||\; B_2 \xrightarrow{a} B_1'}$$

$$\frac{B_1 \;|G|\; B_2 \xrightarrow{a} B_1'}{B_1 \;||\; B_2 \xrightarrow{a} B_1'}$$

ただし, G はすべてのイベント (ゲート) のリスト.

与えられたLOTOSの動作式はLTSの初期状態に対応し，それから導出されたラベル付き遷移の動作式およびアクションが，それぞれ，状態集合およびアクション集合に対応する．ラベル付き遷移は遷移関係と対応する．

有限ラベル付き遷移システム (Finite LTS: FLTS) とは，状態集合 S と ラベル集合 L が有限のLTSである．本論文では，特にことわらない限りLTSは，各状態が初期状態から到達可能なFLTSを意味する．本論文ではLTSをその初期状態によって参照する． また，LTS $TS = (S, L, T, s_0)$ のとき $Act(TS)$ を $L - \{\mathbf{i}\}$ で，$Init(s)$ を $\{e \mid \exists s', (s, e, s') \in T, e \neq \mathbf{i}\}$ で，$Pre(s)$ を $\{e \mid \exists s', (s', e, s) \in T, e \neq \mathbf{i}\}$ で定義する．

2.5 等価性

LOTOS仕様の解析を行うために，双模倣等価と呼ばれる同値関係が定義されている．これは，LOTOSの動作式の形式的解釈であるLTSの上で定義され，外部観測上区別不可能な動作を行うLTSを相互に関係付けるものである．

双模倣等価は，その定義に内部アクションを含むかどうかで二つに分けられる．内部アクションを含めないものを強双模倣等価といい，内部アクションを含めるものを弱双模倣等価という．

本節ではこれらの関係について文献 [3, 9] を要約する．

2.5.1 強双模倣等価

動作式 B_1 と B_2 によって表現される2つのLOTOS仕様の等価性は，B_1 と B_2 それぞれ に以下で定義する遷移規則を適用して得られるLTS間の等価性によって定義される．

定義 2 (アクション系列の遷移関係) (S, L, T, s_0) をLTSとする．

$s, s' \in S$, $a_1, \cdots, a_n \in L - \{\mathbf{i}\}$ とし, t をアクションの系列 $a_1, ..., a_n$ とした時, 遷移関係をアクション系列 t に対して自然に拡張した関係 $\xrightarrow{t}$ を次のように定義する.

$s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n = s'$ であるような状態 $s_0, \cdots, s_n \in S$ が存在する時, これを $s \xrightarrow{t} s'$ で表す. 特に, $n = 0$ に対しては $s \xrightarrow{\varepsilon} s'$ である. ここで ε は空系列を表す.

次に, 関係 $\xrightarrow{t}$ を用いて, 強双模倣関係を定義する.

定義 3 (強双模倣関係) (S_1, L, T_1, s_{01}), (S_2, L, T_2, s_{02}) を LTSs とし, $S = S_1 \cup S_2$ とする. 状態の組 $(s_1, s_2) \in R$ と, 任意の観測可能なイベントの系列 $t \in (L - \{\mathbf{i}\})^*$ に対して, 次の 2 つの条件 a) と b) が成り立つ時, 関係 $R \subseteq S \times S$ を強双模倣関係という.

a) $s_1 \xrightarrow{t} s_1'$ である $s_1' \in S$ が存在するならば, $s_2 \xrightarrow{t} s_2'$ で $(s_1', s_2') \in R$ である $s_2' \in S$ が存在する.

b) $s_2 \xrightarrow{t} s_2'$ である $s_2' \in S$ が存在するならば, $s_1 \xrightarrow{t} s_1'$ で $(s_1', s_2') \in R$ である $s_1' \in S$ が存在する.

定義 4 (状態の強双模倣等価) 2 つの LTSs $(S_1, L, T_1, s_{01}), (S_2, L, T_2, s_{02})$ において, 状態の組 (s_1, s_2) を含むような強双模倣等価関係が存在する時, 状態 s_1 と s_2 は強双模倣等価であるといい, $s_1 \sim s_2$ と書く.

定義 5 (LTS の強双模倣等価) $Sys1 = (S_1, L, T_1, s_{01})$, $Sys2 = (S_2, L, T_2, s_{02})$ を LTSs とし, $S = S_1 \cup S_2$ とする. 状態の組 $(s_{01}, s_{02}) \in R$ であるような強双模倣等価関係 $R \subseteq S \times S$ が存在する時 (すなわち $s_{01} \sim s_{02}$ の時), $Sys1$ と $Sys2$ は強双模倣等価であるといい, $Sys1 \sim Sys2$ と書く.

定義 6 (動作式の強双模倣等価) 2 つの動作式 B_1 と B_2 の LTS をそれぞれ $Sys1$ と $Sys2$ とする. $Sys1 \sim Sys2$ の時, かつその時に限り B_1 と B_2 は強双模倣等価であるといい, $B_1 \sim B_2$ と書く.

2.5.2 弱双模倣等価

弱双模倣等価は, 内部アクションをその定義中に導入している. まず, 前節で定義したアクション系列の遷移関係を, 内部アクションを含んだ関係に拡張する.

定義 7 (アクション系列の遷移関係) (S, L, T, s_0) を LTS とする.

$s, s' \in S$, $a_1, \cdots, a_n \in L - \{\mathbf{i}\}$ とし, t をアクションの系列 $a_1, \cdots, a_n \in (L - \{\mathbf{i}\})^*$ とし, $\mathbf{i}^k$ を k $(k \geq 0)$ 個の内部アクションの系列とした時, 関係 $\stackrel{t}{\Rightarrow}$ を次のように定義する.

$s \stackrel{\mathbf{i}^{k_0} a_1 \mathbf{i}^{k_1} \cdots a_n \mathbf{i}^{k_n}}{\longrightarrow} s'$ であるようなイベントの系列 $\mathbf{i}^{k_0} a_1 \mathbf{i}^{k_1} \cdots a_n \mathbf{i}^{k_n}$ が存在する時, これを $s \stackrel{t}{\Rightarrow} s'$ と表す. 特に, $s \stackrel{\mathbf{i}^k}{\longrightarrow} s'$ を $s \stackrel{\varepsilon}{\Rightarrow} s'$ で表し, また, すべての s に対して $s \stackrel{\varepsilon}{\Rightarrow} s$ とする.

次に, 関係 $\stackrel{t}{\Rightarrow}$ を用いて, 弱双模倣関係を定義する.

定義 8 (弱双模倣関係) (S_1, L, T_1, s_{01}), (S_2, L, T_2, s_{02}) を LTSs とし, $S = S_1 \cup S_2$ とする. 状態の組 $(s_1, s_2) \in R$ と, 任意の観測可能なイベントの系列 $t \in (L - \mathbf{i})^*$ に対して, 次の条件 a) と b) が成り立つ時, 関係 $R \subseteq S \times S$ を弱双模倣関係という.

a) $s_1 \stackrel{t}{\Rightarrow} s_1'$ である $s_1' \in S$ が存在するならば, $s_2 \stackrel{t}{\Rightarrow} s_2'$ で $(s_1', s_2') \in R$ である $s_2' \in S$ が存在する.

b) $s_2 \stackrel{t}{\Rightarrow} s_2'$ である $s_2' \in S$ が存在するならば, $s_1 \stackrel{t}{\Rightarrow} s_1'$ で $(s_1', s_2') \in R$ である $s_1' \in S$ が存在する.

定義 9 (状態の弱双模倣等価) 2 つの LTSs $(S_1, L, T_1, s_{01}), (S_2, L, T_2, s_{02})$ において, 状態の組 (s_1, s_2) を含むような弱双模倣等価関係が存在する時, 状態 s_1 と s_2 は弱双模倣等価であるといい, $s_1 \approx s_2$ と書く.

定義 10 (LTS の弱双模倣等価) $Sys1 = (S_1, L, T_1, s_{01})$, $Sys2 = (S_2, L, T_2, s_{02})$ を LTSs とし, $S = S_1 \cup S_2$ とする. 状態の組 $(s_{01}, s_{02}) \in R$ であるような弱双模倣等価関係 $R \subseteq S \times S$ が存在する時 (すなわち $s_{01} \approx s_{02}$ の時), $Sys1$ と $Sys2$ は弱双模倣等価であるといい, $Sys1 \approx Sys2$ と書く.

定義 11 (動作式の弱双模倣等価) 2 つの動作式 B_1 と B_2 の LTS をそれぞれ $Sys1$ と $Sys2$ とする. $Sys1 \approx Sys2$ の時, かつその時に限り B_1 と B_2 は弱双模倣等価であるといい, $B_1 \approx B_2$ と書く.

2.6 まとめ

本章では, 本研究が基盤としている FDTs について述べた. まず, 2.2 節で FDTs の歴史的背景について述べた. 次に, 2.3 節で形式記述言語 LOTOS について述べ, 2.4 節では, この LOTOS の意味を表現するために使われる LTSs について述べた. 最後に 2.5 節では, 形式記述言語によって記述された仕様間の等価性について述べた.

以上の基盤を使って, 次章以降, 系統的な仕様の分割法・変更法を構成する.

第 3 章

LOTOS 仕様の分割法

3.1 はじめに

本章の目的は, LOTOS 仕様の系統的な分割法を構成することである. この目的を達成するため, 次の 2 項目を提案する.

1) 仕様を構成するアクション集合の分割をもとに, 分割の前後で仕様の動的振舞いの等価性を保つような分割法を構成する.

2) 等価性を保つために, 分割された仕様において同期しなければならないアクション集合を規定した仕様を自動合成する.

まず 3.2 節で分割問題を述べる. 3.3 節で数学的準備を行い, これをもとに 3.4 節では, LOTOS 仕様の分割法を構成する. 3.5 節では仕様の分割例を示す. 3.6 節は本章のまとめである.

3.2 分割問題

分割問題を直観的に示すために, Question/answer service[12] を例に使う.

例 1 (Question/ansewer service) 図 3.1 に示されるように, 質問がユーザ Q によって出され (アクション Qq の生起), このサービスによっ

て別のユーザ A に送られる (Aq). ユーザ A は答えを出すことを要求され (Aa), その答えはこのサービスによってユーザ Q に送られる (Qa).

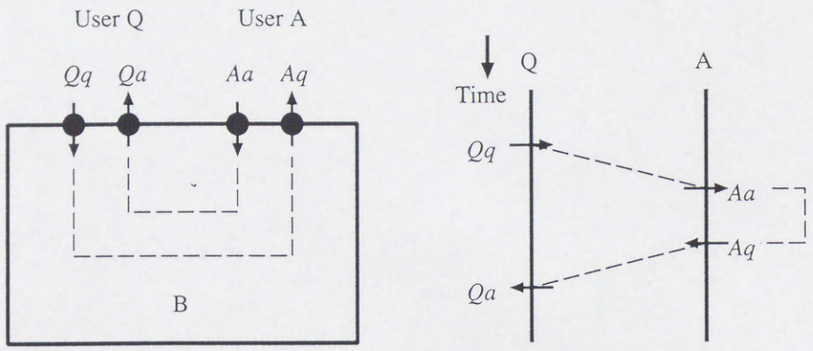


図 3.1: Question/answer service "B"

この非形式的な定義を LOTOS の動作式で直接的に表現すると, 次のモノリシックスタイルのプロセス B が得られる.

$$B := Qq; Aq; Aa; Qa; \mathbf{stop}$$

段階的詳細化の第一ステップとして, まずこの仕様を分割する. そこで以下の 2 つのモジュールに仕様を分割する.

1) ユーザ Q 側だけで生じるアクションから構成される部分 (L_a),

2) ユーザ A 側だけで生じるアクションから構成される部分 (L_b).

これらは, それぞれシステムの Q 側と A 側の局所的な機能である. 従ってお互いに内部で同期をとらない独立なプロセスとして表現される.

一方で, 分割された 2 つのプロセスと分割前のプロセスが, ある等価関係を持つことが必要である. このために, 1 つの並列するプロセス R を付加的に加える.

そこで $\cong$ をある等価関係とすると, Question/answer service 分割問題は次のように定式化される (図 3.2 参照):

Question/answer service 分割問題

以下のような性質を持つ 2 つのプロセス L_a, L_b と, これに付加するプロセス R を見つける:

$$(L_a \,|||\, L_b)\ |G|\ R \cong B$$

ここで, G は 2 つの L と R が同期をとるアクション集合であり, $\{Qq,\ Aq, Aa, Qa\}$ の部分集合である. また, R は n 個の独立並列なプロセスから構成されており:

$$R := R_1 \,|||\, R_2 \,|||\, \cdots \,|||\, R_n \quad (n \geq 1)$$

である.

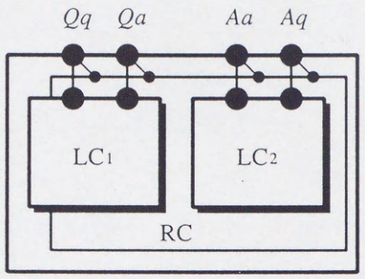


図 3.2: 分割後の Question/answer service "B"

3.3 数学的準備

本章では, 使用する言語として Basic LOTOS を用い, データ型や値式は用いない. 使用するオペレータは Basic LOTOS のうち, (1)**stop**, (2) アクションプレフィックス, (3) チョイス, (4) 並列オペレータの 4 つである. 分割の前後におけるプロセス (すなわち, B, L_a, L_b) と付加的なプロセス R (すなわち, R_1, ..., R_n) は, (1)**stop**, (2) アクションプレ

フィックス, (3) チョイスの 3 つを用いて表現され, 分割後のプロセスが分割前のプロセスと等価になるように, 並列オペレータで分割後のプロセスと R を結合する.

分割されるプロセス B の記述スタイルはモノリシックスタイル [12] であると仮定する. ここで, モノリシックスタイルの標準形は次のように表現される [13].

$$B = \sum\{a_i; B_i \mid i \in I\}$$

ここで, I は有限の添字集合であり, B_i は B と同じ形式を持つプロセスである. $\sum$ はチョイスの一般形を示す.

例えば $\sum\{a_i; B_i \mid i \in \{1,2,3\}\}$ は $a_1; B_1$ [] $a_2; B_2$ [] $a_3; B_3$ を意味する.

集合 S の分割 (partition) とは, $\{S_1, S_2, ..., S_n\}$ で表わされる, S の部分集合の集合であり, $\cup_{i=1}^{n} S_i = S$ かつ $S_i \cap S_j = \emptyset$ $(i \neq j)$ であるものをいう.

次に, 本論文で扱う分割問題を正確に与える.

仕様の分割問題

以下の 2 つの入力が与えられたとする.

1) モノリシックスタイルのプロセス B,

2) $Act(B)$ の分割 $\{A_a, A_b\}$.

この時, 以下の 2 つの条件を満足する 2 つのプロセス L_a と L_b を見つけ, 同時に付加プロセス R を見つけるのがここでの分割問題である.

a) $Act(L_a) = A_a$, $Act(L_b) = A_b$ であり, かつ $Act(R) \subseteq Act(B)$.

b) $(L_a \,|||\, L_b) \,|G|\, R \cong B$

ここで $R := R_1 \,|||\, R_2 \,|||\, \cdots \,|||\, R_n$ $(n \geq 1)$, $G \subseteq Act(B)$

仮定 1 (分割されるプロセスの制限) 本章では, 以下のような制約を持ったプロセスを分割の対象とする.

1) 内部アクション **i** を含まない.

2) 再帰を含まない.

3) 記述されている各アクションは互いに違う名前である.

分割数は 2 としているが, 本アルゴリズムを複数回適用することによって, n 個のプロセスに分割可能である.

3.4 分割法

本節では, 3.3 節で述べた分割問題を解決するアルゴリズムを示す. まず, 3.4.1 節で諸定義を述べ, 3.4.2 節で分割アルゴリズムを示す. 3.4.3 節でアルゴリズムの持つ諸性質について述べる.

3.4.1 諸定義

定義 12 (連結プロセス) あるプロセス B を展開する過程において生じる任意のプロセス B' とアクション集合 A に対し, 次の 2 つの条件のいずれかが成立するとき, B' をアクション集合 A による連結プロセスという.

- $Pre(B')$ が存在せず, かつすべての $a \in Init(B')$ に対して $a \in A$ であるとき.
- $Pre(B') \subseteq A$ であり, かつすべての $a \in Init(B')$ に対して $a \in A$ であるとき.

例えば, $B := a_1; B_1 \ [] \ a_2; B_2$, $B_1 := a_3; B_3 \ [] \ a_4; B_4$ であり, かつ $A = \{a_1, a_2, a_3, a_4\}$ であるとき, B と B_1 は A による連結プロセスである (図 3.3 (a) 参照).

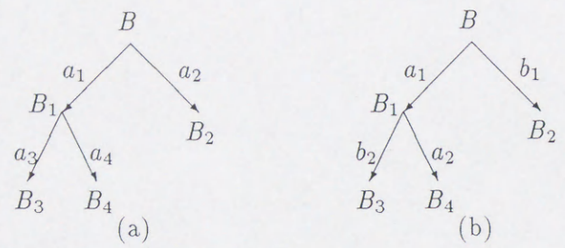


図 3.3: 連結プロセスと切断プロセスの例の LTSs

定義 13 (切断プロセス) あるプロセス B を展開する過程において生じる任意のプロセス B' と，互いに素なアクション集合 A_α と A_β に対し，次の 2 つの条件のいずれかが成立するとき，B' をアクション集合 A_α と A_β による切断プロセスという．

- $Pre(B')$ が存在せず，かつある $a \in Init(B')$ が存在して $a \in A_\alpha$ であり，かつある $b \in Init(B')$ が存在して $b \in A_\beta$ であるとき．
- $Pre(B') \subseteq A_\alpha$ であり，かつある $a \in Init(B')$ が存在して $a \in A_\beta$ であるとき．

例えば，$B := a_1; B_1 \ [] \ b_1; B_2$，$B_1 := b_2; B_3 \ [] \ a_2; B_4$ であり，かつ $A_a = \{a_1, a_2\}, A_b = \{b_1, b_2\}$ であるとき，B と B_1 は A_a と A_b による切断プロセスである (図 3.3 (b) 参照)．

定義 14 (プロセスの再現) あるプロセスに対して，これと同じプレアクションと同じイニシャルを持つような別のプロセスを構成することを，プロセスの再現という．ただし，プレアクションが存在しないプロセスに関しては，プレアクションを持たずかつ同じイニシャルを持つプロセスを構成することを再現という．

例えば，$B := a; B_1$，$B_1 := b; B_2 \ [] \ c; B_3$ があるとする．この時，$C := x; a; C_1$，$C_1 := b; C_2 \ [] \ c; C_3$ を構成したとすると，$Pre(B_1) =$

$Pre(C_1) = \{a\}$, かつ, $Init(B_1) = Init(C_1) = \{b, c\}$ であるので, C_1 は B_1 の再現である (図 3.4 参照).

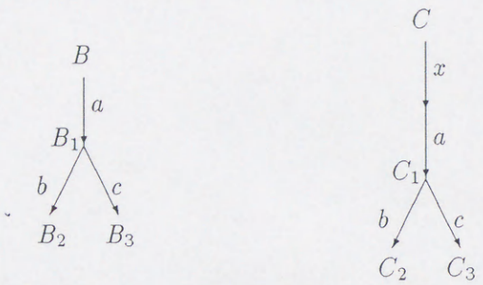


図 3.4: プロセスの再現の例の LTSs

1 つのプロセスから, あるアクション集合の要素だけで構成されるプロセスを抽出するために, 次のような写像 Rest を導入する:

定義 15 (写像 Rest) モノリシックスタイルのプロセス $B := \Sigma\{a_i; B_i \mid i \in I\}$ とアクション集合 A に対して, 写像 $Rest(B, A)$ を次のように定義する.

$$\begin{aligned} Rest(B, A) \;:=\; & \Sigma\{a_i; Rest(B_i, A) \mid a_i \in A, i \in I\} \\ [] \;\; & \Sigma\{Rest(B_i, A) \mid a_i \notin A, i \in I\} \end{aligned}$$

ただし, $Rest(\mathbf{stop}, A) := \mathbf{stop}$ である.

例えば, $B := a; (b; c; \mathbf{stop}\ []\ d; e; \mathbf{stop})\ []\ f; g; h; \mathbf{stop}$ であり, かつ $A = \{a, b, e, h\}$ であるとき, $Rest(B, A) = a; (b; \mathbf{stop}\ []\ e; \mathbf{stop})\ []\ h; \mathbf{stop}$ である. $Rest(B, A)$ の LTS は図 3.5 に示される.

3.4.2 分割アルゴリズム

プロセスの再現性とプロセス間の並列合成関係を考慮し, 写像 Rest を用いて以下の戦略を構成する.

戦略 1

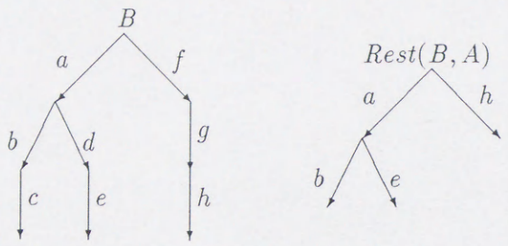


図 3.5: Rest の例の LTSs

1) **L の構成法**

B から $Rest$ を用いて A_a と A_b だけで構成されるプロセスに写像したものを, それぞれ L_a と L_b とする.

2) **連結プロセスと切断プロセスに関する構成法**

B を展開する過程における全てのプロセスに対して,

a) A_a と A_b による連結プロセスを L_a か, あるいは L_b に再現する (これは 1) の構成法によって再現される).

b) A_a と A_b による切断プロセスを R のうちのどれか 1 つに再現する.

3) **R におけるアクションの非冗長性について**

R を構成しているアクション集合 C_k に関して

$$\bigcup_{k\in\{1,2,\ldots,n\}} C_k \subseteq Act(B)$$

$$C_i \cap C_j = \emptyset \ (i \neq j)$$

とする.

4) **L と R の同期アクション集合 G について**

同期アクション集合 G を,

$$G = \bigcup_{k\in\{1,2,\ldots,n\}} C_k$$

とする.

以上の戦略をもとに, 3.3 節で述べた分割問題を解決するアルゴリズムを構成する.

アルゴリズム 1 (LOTOS 仕様の分割)

入力: 分割されるプロセス B と $Act(B)$ の分割 $\{A_a, A_b\}$.

出力: 分割されたプロセス L_a, L_b と R $(R_1 \,|||\, \cdots \,|||\, R_n, n \geq 1)$

(Step 1) R を構成するアクション集合 $C_1, C_2, ..., C_n$ を戦略 1 に基づいて求める.

(Step 1.1) 分割されるプロセス B の展開過程において生じる **stop** を除いたすべてのプロセスに対して, そのイニシャルとプレアクションとの和集合を作る. ただし, プレアクションがないプロセスに関しては, イニシャルだけで 1 つの集合とする. これらの集合に対しては, 作った順番に添字を付ける.

(Step 1.2) ステップ 1.1 で作った集合のうち A_a の要素だけで構成されている集合と, A_b の要素だけで構成されている集合を取り除く.

(Step 1.3) ステップ 1.2 で残った集合に対して, 共通要素を持つ集合どうしの和集合をつくる. この和集合には 2 つの集合のうち添字の小さい方をつけ, 2 つの集合は取り除く. この操作を, 全ての集合が共通要素を持たなくなるまで繰り返す.

(Step 1.4) ステップ 1.3 で残った集合に対して, 集合の添字の小さい順に集合名を $C_1, C_2, ..., C_n$ とする.

(Step 2) L_a, L_b と $R_1, R_2, ...R_n$ をそれぞれ求める. すなわち写像 Rest を使って次のように求める.

$$L_a := Rest(B, A_a),\ L_b := Rest(B, A_b)$$

$$R_1 := Rest(B, C_1),\ ...,\ R_n := Rest(B, C_n)$$

3.4.3 アルゴリズムの性質

アルゴリズム 1 によってプロセスされた L_a, L_b および R は, 元のプロセス B に対して次のような性質を持つ.

性質 1 (連結プロセスと切断プロセスの再現に関して) 分割されるプロセス B を展開する過程における任意のプロセス B' に対して, アルゴリズム適用後のプロセスは, 次のような性質を持つ.

1) B' が A_a と A_b による連結プロセスである場合

この B' におけるものと同じプレアクションと同じイニシャルを持つプロセスが, L_a か L_b のいずれかの展開過程のプロセスに存在する. B' がプレアクションを持たない連結プロセスの場合は, 同じイニシャルを持つプロセスが存在する.

2) B' が A_a と A_b による切断プロセスである場合

この B' におけるものと同じプレアクションと同じイニシャルを持つプロセスが, ある R_k の展開過程のプロセスに存在する. B' がプレアクションを持たない切断プロセスの場合は, 同じイニシャルを持つプロセスが存在する.

(証明) 戦略 1 の 2) に従っているアルゴリズム 1 と, 写像 Rest の定義より明らかである. □

性質 2 (連結プロセスと切断プロセスの再現数に関する性質)

1) 分割されるプロセス B の展開過程における, A_a と A_b によるすべての連結プロセスが, L の展開過程に同じ数だけ再現されており, R には再現されていない.

2) 変換されるプロセス B の展開過程における, A_a と A_b によるすべての切断プロセスが, R の展開過程に同じ数だけ再現されており, L には再現されていない.

(証明) 戦略1の1)から3)の下で構成されているアルゴリズム1と, 写像 Rest の定義より明らかである. □

モノリシックスタイルのプロセス B とアルゴリズム1によって得られたプロセスに対して, 次の定理が成立する.

定理 1 B をモノリシックスタイルのプロセスとし, L_a, L_b と $R_1, R_2, ..., R_n$ がアルゴリズム1で得られたとする. このとき B は,

$$D := (L_a \,|||\, L_b) \,|G|\, (R_1 \,|||\, R_2 \,|||\, \cdots \,|||\, R_n)$$

$$G = \bigcup_{k \in \{1,2,...,n\}} C_k$$

に変換され, $B \approx D$ が成立する.

(証明) 分割されるプロセス B とアルゴリズム適用後のプロセス D とが, 強双模倣等価であることを示すには, B と D とが同じLTSを生成することを示せば十分である.

そのため, あるプロセスに対して, そのプレアクションが生起しないとそのプロセスにはならないことを考慮し, 以下の A) と B) を示せばよい.

A) B と D とが同じイニシャルを持つことを示す.

B) B と D を展開する過程における任意のプロセス B' と D' に対して, B' と D' のプレアクションが同じならば B' と D' は同じイニシャルを持つことを示す.

ここで, B を展開する過程で生じるあるプロセス B' は, 定義12と定義13から, 次の3種類に分けることができる. (a) 連結プロセス, (b) 切断プロセス, (c)**stop**.

(c) の **stop** は無動作プロセスであり, アクションの生起には関与しない. 従って A) と B) を議論する上で, (a) と (b) の場合について議論すればよい.

A) の証明

B は, 定義 12 と定義 13 から, 次の 2 つに分けることができる.

(a) プレアクションを持たない連結プロセス

(b) プレアクションを持たない切断プロセス

(a) B がプレアクションを持たない連結プロセスの場合

B において生じ得る可能性があるアクションは, B のイニシャルの要素のうちの 1 つである. 性質 1 よりこの B のイニシャルと同じイニシャルを持つプロセスが, D の L_a あるいは L_b のどちらかにだけ再現されている. そして, 性質 2 より R には再現されていない.

B において生じ得る可能性があるアクションは, そのアクションが生じた後のプロセスに関して, 定義 12 と定義 13 から次の 3 つに場合分けできる.

(i) 連結プロセスのプレアクションである場合

(ii) 切断プロセスのプレアクションである場合

(iii) **stop** のプレアクションである場合

(i) の場合, 性質 1 より, D において, このアクションは L_a あるいは L_b にだけ存在し, 他のどのアクションとも同期をとらず単独で生起する.

(ii) の場合, 性質 1 より, D において, このアクションは L_a あるいは L_b のどちらかとある 1 つの R_k に存在し, これらが同期をとって生起する.

(iii) の場合は, (i) の場合と同様である.

従って, D における L_a と L_b の独立並列合成, および $R_1, R_2, ..., R_n$ の間の独立並列合成の関係, さらに両者の間の一般並列合成の関係により, B と D は同じイニシャルを持つ.

(b) B がプレアクションを持たない切断プロセスの場合

B において生じ得る可能性があるアクションは, B のイニシャルの要素のうちの1つである. 性質1 よりこの B のイニシャルと同じイニシャルを持つプロセスが, D のある R_k にだけ再現されている. そして, 性質2 より L には再現されていない.

B において生じ得る可能性があるアクションは, そのアクションが生じた後のプロセスに関して, (a) と同様に3つに場合分けできる. しかしすべての場合に対し, D において, このアクションはある R_k と L_a あるいは L_b にだけ存在し, これらが同期をとって生起する.

従って, D における L_a と L_b の独立並列合成, および $R_1, R_2, ..., R_n$ の間の独立並列合成の関係, さらに両者の間の一般並列合成の関係により, B と D は同じイニシャルを持つ.

B) の証明

次に, B と D を展開する過程で生じるあるプロセス B' と D' が同じ名前のプレアクションを持つと仮定する. すると B' は, (a) 連結プロセス, (b) 切断プロセス, (c)**stop** のいずれかの場合に対応する. これらは A) の場合と同様に, D' における各プロセスの間の並列合成の関係により, B' と D' が同じイニシャルを持つことが容易に示せる.

以上より, B と D は同じ LTS を生成し, 定理1 が成立する. □

3.5 適用例

本節では具体的な適用例として, Abracadabra service[1] の仕様を使い, 本章で構成したアルゴリズムを適用する.

Abracadabra service は, 次のような LOTOS のモノリシックプロセスとして記述できる.

```
process Abracadabra_service [ConReq,ConInd,ConRes,ConCnf,
                             DatReq,DatInd,DisReq,DisInd] :noexit:=
  ConReq; ConInd; ConRes; ConCnf;          (* Connection Phase    *)
```

```
   DatReq; DatInd;                          (* Data Transfer Phase *)
   DisReq; DisInd;                          (* Disconnection Phase *)
   stop
endproc
```

分割の基準に相当するアクション集合は次のとおりである．これは，データの送信側と受信側という基準で分割している．

$$A_a = \{ConReq, ConCnf, DatReq, DisReq\}$$

$$A_b = \{ConInd, ConRes, DatInd, DisInd\}$$

以上の項目をアルゴリズム 1 への入力とする．アルゴリズムの適用過程は以下のとおりである．

(Step 1) ここでは，R を構成するアクション集合を得るために，以下の操作を行う:

(Step 1.1) ここでは，以下の集合が得られる:

$$\{ConReq\}_1, \quad \{ConReq, ConInd\}_2,$$
$$\{ConInd, ConRes\}_3, \quad \{ConRes, ConCnf\}_4,$$
$$\{ConCnf, DatReq\}_5, \quad \{DatReq, DatInd\}_6,$$
$$\{DatInd, DisReq\}_7, \quad \{DisReq, DisInd\}_8$$

(Step 1.2) 1, 3, 5 の添字を持つ集合が除かれ，以下の集合だけが残る:

$$\{ConReq, ConInd\}_2, \quad \{ConRes, ConCnf\}_4,$$
$$\{DatReq, DatInd\}_6, \quad \{DatInd, DisReq\}_7,$$
$$\{DisReq, DisInd\}_8$$

(Step 1.3) 添字 6 と 7 の集合で $DatInd$ が，添字 7 と 8 の集合で $DisReq$ が共通しているため，これら 3 つの集合の和集合をつくり，添字を 6 とする:

$$\{ConReq, ConInd\}_2,$$
$$\{ConRes, ConCnf\}_4,$$
$$\{DatReq, DatInd, DisReq, DisInd\}_6$$

(Step 1.4) ここでは, 以下のように集合名がつけられる:

$$\{ConReq, ConInd\}_2 \equiv C_1,$$
$$\{ConRes, ConCnf\}_4 \equiv C_2,$$
$$\{DatReq, DatInd, DisReq, DisInd\}_6 \equiv C_3$$

(Step 2) ここでは, Rest を使って以下のプロセスが得られる:

$$L_a := ConReq; ConCnf; DatReq; DisReq; \mathbf{stop}$$
$$L_b := ConInd; ConRes; DatInd; DisInd; \mathbf{stop}$$
$$R_1 := ConReq; ConInd; \mathbf{stop}$$
$$R_2 := ConRes; ConCnf; \mathbf{stop}$$
$$R_3 := DatReq; DatInd; DisReq; DisInd; \mathbf{stop}$$

以上の過程を経てプロセスが分割される. アルゴリズム適用の結果, 分割されたプロセスは次のとおりである.

```
process La [ConReq,ConCnf,DatReq,DisReq] :noexit :=
   ConReq; ConCnf; DatReq; DisReq; stop
endproc

process Lb [ConInd,ConRes,DatInd,DisInd] :noexit :=
   ConInd; ConRes; DatInd; DisInd; stop
endproc
```

また, 定理 1 により, 以下のプロセスと Abracadabra_service とは強双模倣等価である.

```
process Decomposed_Abracadabra_service [DELTA] :noexit :=
  (
```

```
      La [ALPHA1] (* Sender constraint *)
   |||
      Lb [ALPHA2] (* Receiver constraint *)
   )
   ||
   ( R1 [BETA1] ||| R2 [BETA2] ||| R3 [BETA3] )
where
   (* La と Lb の定義は上記を参照 *)
   process R1 [BETA1] :noexit :=
      ConReq; ConInd; stop
   endproc
   process R2 [BETA2] :noexit :=
      ConRes; ConCnf; stop
   endproc
   process R3 [BETA3] :noexit :=
      DatReq; DatInd; DisReq; DisInd; stop
   endproc
endproc

(* DELTA = ConReq,ConInd,ConRes,ConCnf,DatReq,DatInd, *)
(*         DisReq,DisInd                              *)
(* ALPHA1= ConReq,ConCnf,DatReq,DisReq                *)
(* ALPHA2= ConInd,ConRes,DatInd,DisInd                *)
(* BETA1 = ConReq,ConInd                              *)
(* BETA2 = ConRes,ConCnf                              *)
(* BETA3 = DatReq,DatInd,DisReq,DisInd                *)
```

記述スタイルの観点では, プロセス Decomposed_Abracadabra_service は, モノリシックスタイルの Abracadabra_service を制約指向スタイルに変換したものに相当する.

以上のプロセスの LTSs は, 図 3.6 に示される.

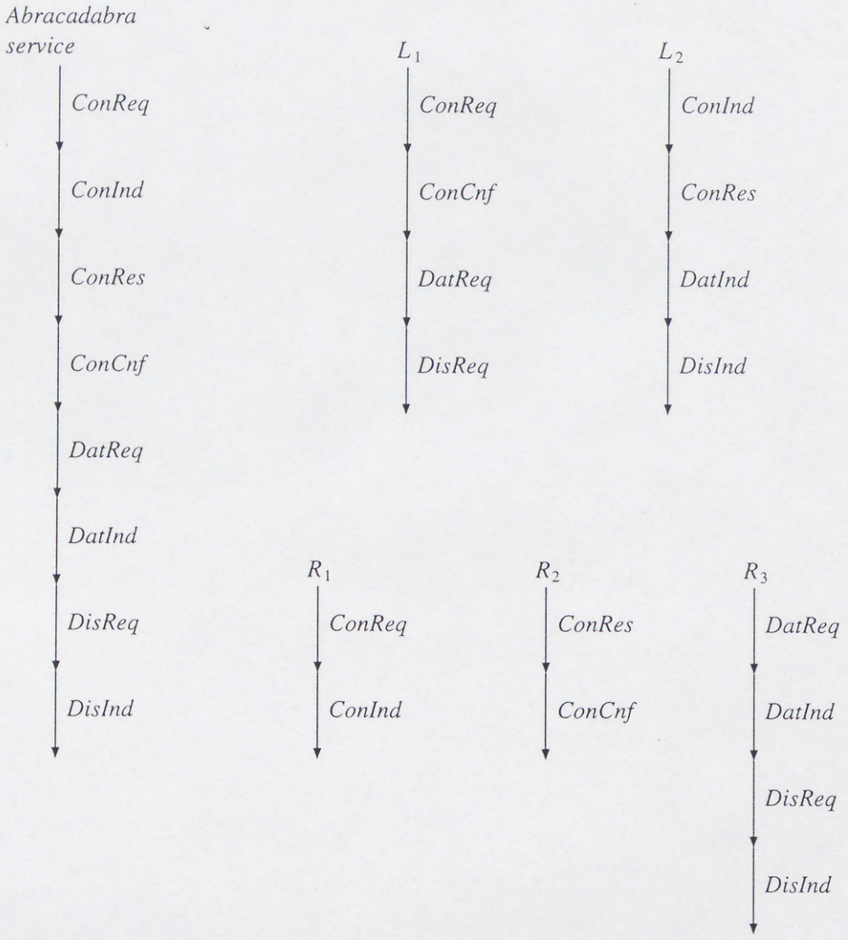


図 3.6: Abracadabra_service 分割における LTSs.

3.6 まとめ

本章では, LOTOS 仕様の系統的な分割法を構成することを目的に次の 2 項目を提案した.

1) 仕様を構成するアクション集合の分割をもとに, 分割の前後で仕様の動的振舞いの等価性を保つような分割法を構成した.

2) 等価性を保つために, 分割された仕様において同期しなければならないアクション集合を規定した仕様を自動合成した.

分割法が分割の前後において仕様の等価性を保つということは, 分割作業によって望ましくないエラーを仕様中に加えることがないことを意味する. この観点から, 本分割法は高品質・高信頼なシステムの仕様を段階的に詳細化するうえで有効な分割法である.

一方で, より一般的な仕様の分割法の構成を目指すには, LOTOS 以外の枠組で同様の分割法が構成できることを示すことと, 仮定 1 で与えた分割可能な仕様に対する制限を取り除く必要がある.

第 4 章

LTS 仕様の分割法

4.1 はじめに

第 3 章では系統的な仕様の分割法を構成することを目的に, LOTOS 仕様の分割法を構成した. しかしながら, この分割法は実用的観点から, 次の制限を持っている.

1) LOTOS という仕様記述言語の枠内でだけ使用可能な分割法である.

2) 分割法が分割可能な仕様に対して次の制限を持っている.

   a) 再帰を含まない.

   b) 内部アクション **i** を含まない.

   c) 仕様に記述されている各アクションは互いに違う名前である.

以上の観点から本章の目的は次の 2 点である.

1) 第 3 章で構成した分割法を一般化し, 任意のプロセス代数仕様に適用できるようにする.

2) 第 3 章で構成した分割法で分割可能な仕様の領域を広げる.

以上の目的を達成するため, プロセス代数仕様の意味表現である LTS に対して, 等価性を保つ分割法を構成する.

4.2 数学的準備

本節では, 本章で用いる数学的表現を定義する.

まず, 2 つの LTSs の並列合成を定義する. この並列合成オペレータは, 2 つのシステムの並行動作の結果として得られる全体の動作を表現するために使われる.

定義 16 (2 つの LTSs の並列合成) 次の 2 つの LTSs $TS_1 = (S_1, L_1, T_1, s_{10}), TS_2 = (S_2, L_2, T_2, s_{20})$ の同期アクション集合 A による並列合成 $TS_1 \,|A|\, TS_2$ を次の LTS で定義する.

$$(S_p, L_1 \cup L_2, T_p, (s_{10}, s_{20}))$$

ここで $S_p \subseteq S_1 \times S_2$ は次の遷移関係 T_p を満たす要素で構成された最小の集合; $T_p \subseteq [S_1 \times S_2] \times [L_1 \cup L_2] \times [S_1 \times S_2]$ は次を満たす要素で構成された最小の集合で定義される.

$(s_1, e, s_1') \in T_1$ かつ $e \notin A$ ならば $((s_1, s_2), e, (s_1', s_2)) \in T_p$,
$(s_2, e, s_2') \in T_2$ かつ $e \notin A$ ならば $((s_1, s_2), e, (s_1, s_2')) \in T_p$,
$(s_1, e, s_1') \in T_1$ かつ $(s_2, e, s_2') \in T_2$ かつ $e \in A$ ならば
$((s_1, s_2), e, (s_1', s_2')) \in T_p$.

省略記法として, $TS_1 \,|\emptyset|\, TS_2$ を $TS_1 \,|||\, TS_2$ と記述し, $TS_1 \,|A_1 \cup A_2|\, TS_2$ を $TS_1 \,||\, TS_2$ と記述する.

次に, 状態集合 S を持つ LTS に対して, あるアクション集合 A に関する S 上の 2 項関係 R_A を次のように定義する.

定義 17 状態集合 S を持つ LTS のある状態 $s_i, s_j \in S$ に対し, $(s_i, s_j) \in R_A$ であるとは, 次の 2 つの条件が成立するときかつそのときに限る.

1. $\exists \sigma, \sigma' \in A^*,\ s_i \xrightarrow{\sigma} s_j \vee s_j \xrightarrow{\sigma'} s_i$;

2. $\exists s_k \in S,\ \exists \sigma, \sigma' \in A^*,\ s_k \xrightarrow{\sigma} s_i \wedge s_k \xrightarrow{\sigma'} s_j.$

上で定義した関係 R_A は同値関係であり，縮退同値関係 (reduced equivalence) と呼ばれる．R_A が同値関係であることは次のことからわかる．

反射律: 任意の $s_i \in S$ は A の要素による 0 回の遷移で s_i に到達可能であるから，$(s_i, s_i) \in R_A$ である．

対称律: 定義の 1. から，任意の $s_i, s_j \in S$ に対して，$(s_i, s_j) \in R_A$ ならば，$(s_j, s_i) \in R_A$ である．

推移律: 定義の 1. と 2. から，任意の $s_i, s_j, s_k \in S$ に対して，$(s_i, s_j) \in R_A$ かつ $(s_j, s_k) \in R_A$ ならば，$(s_i, s_k) \in R_A$ である．

例 2 LTS $B = (S, L, T, s_0)$ を仮定する．ここで，

$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$;
$L = \{a, b, c, d, e, f, g, h\}$;
$T = \{(s_0, a, s_1), (s_1, b, s_2), (s_2, c, s_3), (s_0, d, s_4), (s_4, e, s_5), (s_5, f, s_6), (s_4, g, s_7), (s_7, h, s_8)\}$.

B の $A = \{a, b, e, g\}$ に関する S 上の縮退同値関係 R_A は次のとおりである．

$R_A = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_1), (s_1, s_2), (s_2, s_0), (s_2, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4), (s_4, s_5), (s_4, s_7), (s_5, s_4), (s_5, s_5), (s_5, s_7), (s_6, s_6), (s_7, s_4), (s_7, s_5), (s_7, s_7), (s_8, s_8)\}$.

LTS (S, L, T, s_0) に対して，アクション集合 A に関する S 上の縮退同値関係 R_A から，S を分割することが可能である．S の部分集合 $[a] = \{b \mid b\ R_A\ a\}$ を R_A の同値類 (equivalence class) と呼ぶ．S に関する R_A のすべての同値類の集合 π_A を S の縮退分割 (reduced partition) と呼ぶ．

例 3 例 2 の B と R_A に対して, R_A による S の縮退分割 π_A は

$$\{[s_0], [s_3], [s_4], [s_6], [s_8]\}$$

である. ここで,

$$[s_0] = \{s_0, s_1, s_2\}; [s_3] = \{s_3\}; [s_4] = \{s_4, s_5, s_7\};$$
$$[s_6] = \{s_6\}; [s_8] = \{s_8\}.$$

ここで, LTS の状態を状態集合の同値類に拡張し, これによって LTS を定義する. 以下で定義する縮退プロセス (reduced process) B/π_A は, アクション集合 $Act(B) - A$ の要素による遷移によって構成されたプロセスである.

定義 18 $B = (S, L, T, s_0)$ を LTS とし, R_A を A による S 上の縮退同値関係とする. R_A によって導出された S の縮退分割を $\pi_A = \{[s] \mid s \in S\}$ とする. π_A に関する縮退プロセス B/π_A を, LTS $(\pi_A, L - A, T', [s_0])$ で定義する. ここで, もし $([s_i], a, [s_j]) \in T'$ かつそのときに限り $\exists s'_i \in [s_i], s'_j \in [s_j], (s'_i, a, s'_j) \in T$.

例 4 上記の例で, π_A に関する B の縮退プロセス B/π_A は $(\pi_A, L - A, T', [s_0])$ である. ここで

$$\pi_A = \{[s_0], [s_3], [s_4], [s_6], [s_8]\};$$
$$L - A = \{c, d, f, h\};$$
$$T' = \{([s_0], c, [s_3]), ([s_0], d, [s_4]), ([s_4], f, [s_6]), ([s_4], h, [s_8])\}.$$

4.3 分割法

本節では, 分割法を 3 段階に分けて構成する. 一段階につき, 入力仕様に対するひとつの制限を取り除き, 最終的に, すべての制限を取り除いた仕様に対する分割法を構成する. 各段階は次のとおりである.

段階 1：入力するプロセス B は次の 2 つの制限を持つ

a) B は内部アクション $\mathbf{i}$ を含まない.

b) B に記述されている各アクションは互いに違う名前である.

段階 2：制限 b) を取り除く.

段階 3：B に対するすべての制限を取り除く.

4.3.1 段階 1

段階 1 のアルゴリズムに対する入力プロセス B は, 次の 2 つの制限を持ったプロセスである.

a) B は内部アクション $\mathbf{i}$ を含まない.

b) B に記述されている各アクションは互いに違う名前である.

本節のアルゴリズムを構成するために, 2 つの副アルゴリズムを準備する.

副アルゴリズム 1 は, プロセス B とアクション集合 A を入力とし, B から A の要素だけで構成されるプロセスを抽出する役目をはたす. このために, 副アルゴリズム 1 は B を構成する状態集合 S の A にもとづく分割 π_A を構成する.

副アルゴリズム 2 は, プロセス B と互いに素な 2 つのアクション集合 A_a と A_b を入力とする. アルゴリズムは B の任意の状態 s_i に対して, それが関与するすべての遷移のラベルから構成される集合 A_i を作る. もし A_i が A_a と A_b の両方の要素から構成されている場合, A_i を A の要素に加える操作を行う.

以下, 順に副アルゴリズムを示す.

副アルゴリズム 1

procedure $Rest(B, A)$:

入力: $B = (S, L, T, s_0)$, and A.

出力: $\mathcal{S}$.

begin

$\mathcal{S} \leftarrow \{\{s\} \mid s \in S\}$;

for each $(s, e, s') \in T$

　　if $a \notin A$ **then**

　　　　$\mathcal{S} \leftarrow (\mathcal{S} - \{[s], [s']\}) \cup \{[s] \cup [s']\}$;

return $\mathcal{S}$

end.

次に, 副アルゴリズム 2 を以下に示す. 副アルゴリズム 2 中では, アクション集合を互いに素にするため, 副アルゴリズム 3 を呼ぶ.

副アルゴリズム **2**

procedure $MakeSetR(\mathcal{S})$:

入力: $B = (S, L, T, s_0)$, A_a, A_b, ここで $A_a \cup A_b = L, A_a \cap A_b = \emptyset$.

出力: $\mathcal{A} = \{A_i \mid i = 1, 2, ..., n\}$.

$n \leftarrow 1$;

$A_1 \leftarrow Init(s_0)$;

$\mathcal{A} \leftarrow \{A_1\}$;

for each $s_i \xrightarrow{a} s_j \in T$

　　begin

　　　　$n \leftarrow n + 1$;

　　　　$A_n \leftarrow \{a\} \cup Init(s_j)$;

　　　　$\mathcal{A} \leftarrow \mathcal{A} \cup \{A_n\}$;

　　end

for each $A_i \in \mathcal{A}$

if $A_i \subset A_a$ or $A_i \subset A_b$ **then**

$\mathcal{A} \leftarrow \mathcal{A} - \{A_i\}$;

$\mathcal{A} \leftarrow PWDJ(\mathcal{A})$;

return $\mathcal{A}$

end.

副アルゴリズム 3 は, 入力された集合の集合に対し, その要素である集合を互いに素にする処理を行ない, 結果として, 互いに素な集合を要素を持った集合を返す.

副アルゴリズム **3**

procedure $PWDJ(\mathcal{S})$:

入力: $\mathcal{S} = \{S_i \mid i = 1, 2, ..., n\}$,

出力: $\mathcal{S}$.

begin

while $\exists S_i, S_j \in \mathcal{S}\ (i \neq j), S_i \cap S_j \neq \emptyset$ **do**

$\mathcal{S} \leftarrow (\mathcal{S} - \{S_i, S_j\}) \cup \{S_i \cup S_j\}$;

return $\mathcal{S}$.

end.

以上の副アルゴリズムを使用し, 段階 1 のアルゴリズムは次のように示される.

アルゴリズム **2**

入力: B, $Act(B)$ の分割 $\{A_a, A_b\}$.

出力: L_a, L_b, ここで $Act(L_a) = A_a, Act(L_b) = A_b$. $R_i\ (i = 1, 2, ..., n)$, ここで各 $Act(R_i)$ は互いに素.

(Step 1) 副アルゴリズム 2 を B, A_a, A_b に適用し $A_1, A_2, ..., A_n$ を得る.

(Step 2) (L_a, L_b, R の構成):

各 $\alpha = a, b$ に対し, 副アルゴリズム 1 を B と A_α に適用し, π_α を得る. ここから $L_\alpha := B/\pi_\alpha$ を構成する.

各 $i = 1, 2, ..., n$ に対し, 副アルゴリズム 1 を B と A_i に適用し, π_i を得る. ここから $R_i := B/\pi_i$ を構成する.

以下本論文では, B を任意のプロセスとし, $B' = (L_a \ ||| \ L_b) \ |G| \ (R_1 \ ||| \ R_2 \ ||| \ ... \ ||| \ R_n)$ を B を分割して得られたサブプロセスの並列合成とする. ここで, $G = \cup_{i=1}^{n} Act(R_i)$ は同期アクションの集合である.

B の状態を $(n+2)$ 項組 $(s_a, s_b, s_1, ..., s_n)$ で示す. ここで, s_a, s_b, s_i はそれぞれ L_a, L_b, R_i の状態である.

アルゴリズム 2 から構成されたプロセスは, 以下の性質を持つ.

補題 1 $Act(R_1), Act(R_2), ..., Act(R_n)$ は互いに素である. □

B' がとる任意の状態は, $([s_a]_a, [s_b]_b, [s_1]_1, ..., [s_n]_n)$ で表現できる. ここで, $s_a, s_b, s_1, ..., s_n$ は B の状態であり, $[s_a]_a, [s_b]_b, [s_i]_i$ はそれぞれ π_a, π_b, π_i に関する同値類, π_a, π_b, π_i は B からそれぞれ L_a, L_b, R_i を構成するために使われた分割である.

補題 2 s を B の状態とする. B' において状態 $\tilde{s}$ が存在し, $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} \tilde{s}$ ならば, B において状態 s' が存在し, $\tilde{s} = ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ である.

(証明) $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} \tilde{s}$ を仮定する. B' の構成から, a は L_a か L_b かのどちらか一方で生起する. 一般性を失わず, a が L_a で生起すると仮定する (以下の主張は a が L_b で生起する場合も同様に示せる). すると, L_a において, ゆえに B において, $s \xrightarrow{a} s'$ であるような $s', s_b, s_1, ..., s_n$ が存在し, $\tilde{s} = ([s']_a, s_b, s_1, ..., s_n)$ である.

a は L_b で生起しないので, L_b の構成から $s_b = [s]_b = [s']_b$ である.

証明は, a が $G = \bigcup_{i=1}^{n} Act(R_i)$ の要素かであるかそうでないかで, 次の 2 つの場合に分けられる.

1. $a \in G$. このときある i に対して R_i で $[s]_i \xrightarrow{a} [s']_i$. ゆえに補題 1 と R_i の構成より $\tilde{s} = ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ が得られる.

2. $a \notin G$. このとき a はどの R_i においても生起しない. ゆえに $\tilde{s} = ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ が得られる. □

補題 3 s と s' を B の状態とする. B で $s \xrightarrow{a} s'$ のとき, かつそのときに限り B' で $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} ([s']_a, [s']_b, [s']_1, ..., [s']_n)$.

(証明) ($\Longrightarrow$) この方向は補題 2 と同様に証明できるため省略する.

($\Longleftarrow$) $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ を仮定する. 補題 2 から, B において $s \xrightarrow{a} s'$ は明らかである. □

以上の補題から, 次の定理を得る.

定理 2 B を次の 2 つの制限を持ったプロセスとする. a) 内部アクションを含まない, b) 記述されている各アクション名は互いに違う名前を持つ. $\{A_a, A_b\}$ を $Act(B)$ の分割とする. L_a, L_b, R_i $(i = 1, 2, ..., n)$ をアルゴリズム 2 に B, A_a, A_b を入力して得られたプロセスとする. このとき次式が成立する.

$$B \sim (L_a \,|||\, L_b) \,|G|\, (R_1 \,|||\, ... \,|||\, R_n).$$

(証明) $\mathcal{R} = \{(s, ([s]_a, [s]_b, [s]_1, ..., [s]_n)) \mid s \text{ は} B \text{ の状態}\} \subseteq B \times B'$ とし, $\mathcal{R}$ が強双模倣であることを示せば十分である.

まず, $(s_0, ([s_0]_a, [s_0]_b, [s_0]_1, ..., [s_0]_n)) \in \mathcal{R}$ である. ここで, $([s_0]_a, [s_0]_b, [s_0]_1, ..., [s_0]_n)$ は B' の初期状態である.

もし $s \xrightarrow{a} s'$ ならば, 補題3より
$([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ であるような状態 $([s']_a, [s']_b, [s']_1, ..., [s']_n)$ が存在し, $(s', ([s']_a, [s']_b, [s']_1, ..., [s']_n)) \in \mathcal{R}$.

もし $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ ならば, 補題3より $s \xrightarrow{a} s'$ であるような状態 s' が存在し, $(s', ([s']_a, [s']_b, [s']_1, ..., [s']_n)) \in \mathcal{R}$.

以上から, $B \sim (L_a \,|||\, L_b)\ |G|\ (R_1 \,|||\, ... \,|||\, R_n)$ が成立する. □

例 5 図4.1(a)に示すプロセス B を分割する. アクション集合 $Act(B) = \{a_1, a_2, a_3, b_1, b_2, b_3, b_4\}$ を $A_a = \{a_1, a_2, a_3\}$ と $A_b = \{b_1, b_2, b_3, b_4\}$ に分割すると仮定する. これらをアルゴリズム2に適用すると図4.1(b)に示すプロセスが得られる. 定理2により, $B \sim (L_a \,|||\, L_b) \,||\, (R_1 \,|||\, R_2)$ が成立する.

4.3.2 段階 2

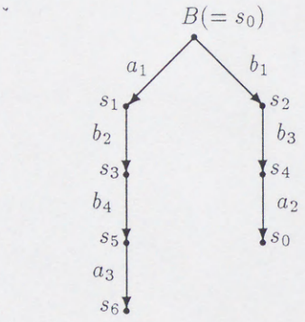
段階1のアルゴリズムでは, 次の2つの制限を持ったプロセス B に対して分割法を構成した.

a) B は内部アクション **i** を含まない.

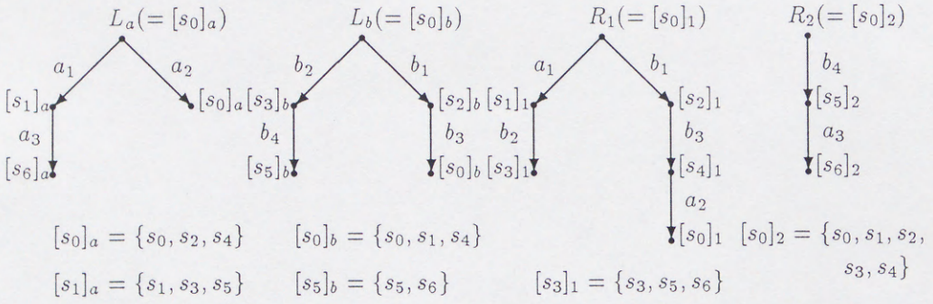
b) B に記述されている各アクションは互いに違う名前である.

段階2では, このうち b) の制限を取り除いたプロセスに対する分割アルゴリズムを構成する. この制限を取り除くということを次のことを意味する. すなわち, 分割されるプロセスにあるアクション名が2回以上出現してもよいということである.

あるアクション名が少なくとも2回現われるようなプロセス B にアルゴリズム2を適用して分割されたプロセスの並列合成 B' は, 望ましくない非決定的アクションが生じるため, 一般に, B と強双模倣等価に



(a) 分割前のプロセス B



(b) アルゴリズム 2 によって分割されたプロセス

図 4.1: 例題 5 における LTSs.

ならない．そのため本節のアルゴリズムでは，B を分割するにあたり次の戦略をとる．まず，B にアルゴリズム 2 を適用し，分割されたプロセスを得る．次に望ましくない非決定的アクションを決定的アクションにするために，分割されたプロセスに対して，プロセスの書換えルールを適用する．

本章のアルゴリズムを構成する上で，以下の準備が必要となる．まず，簡略記法として，$Suc(s,a) = \{s_1, s_2, ..., s_n\}$ のとき $s \xrightarrow{a} s_1 + s_2 + ... + s_n$ と書く．

定義 19 プロセス P において $s \xrightarrow{a} s_1 + s_2 + ... + s_n$ とする．

1. もし $n \leq 1$ ならば，s は a に関して決定的であるといい，$Det(s,a)_P$ と書く．

2. もし $n \geq 2$ ならば，s は a に関して非決定的であるといい，$Ndet(s,a)_P$ と書く．

文脈から P が明らかな場合，添字 P を省略する．

命題 1 L_a, L_b をアルゴリズム 2 に B, A_a, A_b を入力して得られたプロセスとする．このとき，$Ndet(s,a)_B$ ならば，B の任意の s_i と $Act(B)$ の任意の a_i に対し，ある $[s]_a$ が存在して $Ndet([s]_a, a)_{L_a}$，あるいは ある $[s]_b$ が存在して $Ndet([s]_b, a)_{L_b}$． □

命題 2 R_i において $Ndet([s]_i, a)$，かつ R_j において $Ndet([s]_j, a)$ ならば，$i = j$． □

この命題により，$Ndet([s]_i, a)$ がある i によって唯一決定付けられることがわかる．L においても同様に $Ndet([s]_\alpha, a)$ がある α によって唯一決定付けられる．

命題 3 L_a, L_b をアルゴリズム 2 に B, A_a, A_b を入力して得られたプロセスとする．このとき，$Det(s,a)_B$ かつ $Ndet([s]_\alpha, a)_\alpha$ ならば，B の任意の s_i と $Act(B)$ の任意の a_i に対して，$\forall[s]_i, \neg Ndet([s]_b, a)_{L_b}$.

(証明) s を B の任意の状態，a を $Act(B)$ の任意のアクションとする．B において $Det(s,a)$，L_α において $Ndet([s]_\alpha, a)$ を仮定する．$Det(s,a)$ かつ $Ndet([s]_\alpha, a)$ ならば，$s' \neq s$ かつ $\{a\} \subseteq Init(s')$ であるような $s' \in [s]_\alpha$ が B に存在する．R_i の構成から，R_i において $[s] \neq [s']$ かつ $\{a\} \subseteq Act(R_i)$．これは，アルゴリズムが $[s]$ と $[s']$ とを，$[s]$ の直前のアクション，あるいは $[s']$ の直前のアクション，あるいはそれらの両方によって区別するからである．

以上から，$Det(s,a)_B$ かつ $Ndet([s]_\alpha, a)_\alpha$ ならば，$\forall[s]_i, \neg Ndet([s]_b, a)_{L_b}$ が成立する． □

B の状態 s，および L_a, L_b, R の状態 $[s]$ の組合せは，表 4.1 のように場合分けできる．この場合分けをもとに，非決定的状態のための再構成手続きは，次のとおりである．

副アルゴリズム 4

procedure $Rewrite(s,a)$:

入力: プロセスの状態 s.

出力: アクション a.

begin

Case 1 : $Ndet(s,a), Ndet([s]_\alpha, a), Ndet([s]_i, a)$.

$Ndet([s]_\alpha, a)$ から，B に $n \geq 2$ であるような $s \xrightarrow{a} s_1 + s_2 + \ldots + s_n$ が存在する．ある $m \geq 0$ に対して L_α で $[s]_\alpha \xrightarrow{a} [s_1]_\alpha + [s_2]_\alpha + \ldots + [s_n]_\alpha + [s_{n+1}]_\alpha + [s_{n+2}]_\alpha + \ldots + [s_{n+m}]_\alpha$ を仮定する．

1. L_α において，分割 π_α を次のように再構成する: $[s]_\alpha \xrightarrow{a} [s_1]_\alpha + [s_2]_\alpha + \ldots + [s_n]_\alpha + [s_{n+1}]_\alpha + [s_{n+2}]_\alpha + \ldots + [s_{n+m}]_\alpha$ を $[s]_\alpha \xrightarrow{a}$

表 4.1: 状態の性質の分類

	B	L_α	R_i	存在の有無
Case 1	$Ndet(s,a)$	$Ndet([s]_\alpha,a)$	$Ndet([s]_i,a)$	有り
Case 2	$Ndet(s,a)$	$Ndet([s]_\alpha,a)$	$Det([s]_i,a)$	有り
Case 3	$Ndet(s,a)$	$Det([s]_\alpha,a)$	$Ndet([s]_i,a)$	無し (命題 1 から)
Case 4	$Ndet(s,a)$	$Det([s]_\alpha,a)$	$Det([s]_i,a)$	無し (命題 1 から)
Case 5	$Det(s,a)$	$Ndet([s]_\alpha,a)$	$Ndet([s]_i,a)$	無し (命題 3 から)
Case 6	$Det(s,a)$	$Ndet([s]_\alpha,a)$	$Det([s]_i,a)$	有り
Case 7	$Det(s,a)$	$Det([s]_\alpha,a)$	$Ndet([s]_i,a)$	有り
Case 8	$Det(s,a)$	$Det([s]_\alpha,a)$	$Det([s]_i,a)$	有り

$[s_1]_\alpha + [s_{n+1}]_\alpha + [s_{n+2}]_\alpha + ... + [s_{n+m}]_\alpha$ に構成する. ここで $\pi_\alpha \leftarrow (\pi_\alpha - \{[s_1]_\alpha, ..., [s_n]_\alpha\}) \cup \{[s_1]_\alpha \cup ... \cup [s_n]_\alpha\}$.

2. R の再構成: $Ndet([s]_i,a)$ から, $a \in A_j$ であるような $A_j \in \mathcal{A}$ が唯一存在する. まずすべての $i = 1,2,...,n$ に対して, $A_j \leftarrow A_j \cup Init(s_i)$ とする. 次に $\mathcal{A}$ に対して副アルゴリズム 3 を適用する. 最後に各 $A_i \in \mathcal{A}$ に対して副アルゴリズム 1 を適用し, R を導出する.

Case 2 : $Ndet(s,a), Ndet([s]_\alpha,a), Det([s]_i,a)$.

この場合何も行わない.

Case 6 : $Det(s,a), Ndet([s]_\alpha,a), Det([s]_i,a)$.

この場合, L_α で $[s]_\alpha \xrightarrow{a} [s_1]_\alpha + ... + [s_n]_\alpha$ を仮定する. 次に a の非決定性を取り除くために Case 1 の手続きを実行する.

Case 7 : $Det(s,a), Det([s]_\alpha,a), Ndet([s]_i,a)$.

$Ndet([s]_i, a)$ を仮定する．その構成より i は唯一決定できる．仮定から $a \in A_j$ であるような $A_j \in \mathcal{A}$ を唯一決定できる．まず $\mathcal{A}$ のすべての $s' \in [s]_i$ に対して $A_j \leftarrow A_j \cup Pre(s')$ を実行する．結果として得られた $\mathcal{A}$ に副アルゴリズム 3 を適用する．最終的に Case 1 の手続きを実行し，R を導出する．

Case 8 : $Ndet(s, a), Ndet([s]_\alpha, a), Ndet([s]_i, a)$.

この場合何も行わない．

end

以上の再構成手続きを使って，本段階の分割アルゴリズムは，以下のように構成できる．

アルゴリズム 3

入力: B, A_a, A_b.

出力: L_a, L_b, R_i $(i = 1, 2, ..., n)$.

(Step 1) B, A_a, A_b に対してアルゴリズム 2 を適用し，L_a, L_b, R を得る．

(Step 2) B の状態 s とアクション a に対して，B, L_α, R_i が Case 1, Case 6, Case 7 のいずれかに相当する間，副アルゴリズム 4 を適用する．

アルゴリズム 3 によって構成されたプロセスは，以下の性質を持つ．

補題 4 B をプロセスとし，$\{A_a, A_b\}$ を $Act(B)$ の分割とする．L_a, L_b, $R_i (i = 1, 2, ..., n)$ を，アルゴリズム 3 に B, A_a, A_b を入力して得られたプロセスとする．このとき，

1. B の任意の s と $Act(B)$ の任意の a に対して，$Ndet(s, a)$ ならば，$Ndet([s]_\alpha, a)$ かつ $Det([s]_i, a)$，あるいは $Det([s]_\alpha, a)$ かつ $Det([s]_i, a)$.

2. B の任意の s と $Act(B)$ の任意の a に対して, $Det(s,a)$ ならば $Det([s]_\alpha, a)$ かつ $Det([s]_i, a)$. □

補題 4 を表としてまとめたものを, 表 4.2 に示す. 表中で用いられている Case は, 副アルゴリズム 4 および表 4.1 で用いられた Case に対応する.

表 4.2: アルゴリズム 3 適用後における各プロセスの状態の性質

	B	L_α	R_i	存在の有無
Case 1	$Ndet(s,a)$	$Det([s]_\alpha, a)$	$Ndet([s]_i, a)$	有り
Case 2	$Ndet(s,a)$	$Ndet([s]_\alpha, a)$	$Det([s]_i, a)$	有り
Case 3	–	–	–	無し
Case 4	–	–	–	無し
Case 5	–	–	–	無し
Case 6	$Det(s,a)$	$Det([s]_\alpha, a)$	$Det([s]_i, a)$	有り
Case 7	$Det(s,a)$	$Det([s]_\alpha, a)$	$Det([s]_i, a)$	有り
Case 8	$Det(s,a)$	$Det([s]_\alpha, a)$	$Det([s]_i, a)$	有り

補題 5 s を B の状態とする. B' においてある $\tilde{s}$ が存在し, $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} \tilde{s}$ を仮定する. このとき, B で $s \xrightarrow{a} s'$ であるような状態 s' が存在し, $\tilde{s} = ([s']_a, [s']_b, [s']_1, ..., [s']_n)$ である.

(証明) $([s]_a, [s]_b, [s]_1, ..., [s]_n) \xrightarrow{a} \tilde{s}$ を仮定する. B' の a は L_a か L_b かのどちらかで生起する. 一般性を失わず, a が L_a で生起すると仮定する. (以下の主張は a が L_b で生起する場合も同様の方法で示せる). 仮定から, L_a に $s \xrightarrow{a} s'$ であるような状態 $s', s_b, s_1, ..., s_n$ が存在して $\tilde{s} = ([s']_a, s_b, s_1, ..., s_n)$. このゆえに s' は B に存在する. L の構成から, a は L_b で生起せず $s_b = [s]_b = [s']_b$.

s と $[s]_a$ に関して, $Ndet(s,a)$ と $Det(s,a)$ の 2 つの場合がある.

1. $Ndet(s,a)$. この場合はさらに2つの場合に分けられる.

   a) $Ndet([s]_\alpha,a)$ かつ $Det([s]_i,a)$. この場合, a はどの R_i でも生起しない. もし a がある R_i で生起するならば, その構成から $Ndet([s]_i,a)$ となり, これは仮定に矛盾する. ゆえに $a \notin G$. これは, 任意の i に対して, $[s]_i = [s']_i$ を導く. ゆえに, $\tilde{s} = ([s']_a,[s']_b,[s']_1,...,[s']_n)$ が成立する.

   b) $Det([s]_\alpha,a)$ かつ $Ndet([s]_i,a)$. この場合, 仮定と R_i の構成から, L_a の遷移 $[s]_a \xrightarrow{a} [s']_a$ に対応する R_i の遷移 $[s]_i \xrightarrow{a} [s']_i$ が存在しなければならない. 補題2と同様の議論で, この場合において任意の $j \neq i$ に対して $[s]_j = [s']_j$ が成立する. このことから L_a と R_i が a で同期をとり, それぞれ状態 $[s']_a$ と $[s]_i$ に進む. ゆえに $\tilde{s} = ([s']_a,[s']_b,[s']_1,...,[s']_n)$ が成立する.

2. $Det(s,a)$. この場合アルゴリズムから L_α と R_i に対して, $Det([s]_\alpha,a)$ かつ $Det([s]_i,a)$. この証明は補題2と同様の方法で行なえるため省略する. □

補題 6 s と s' を B の状態とする. B で $s \xrightarrow{a} s'$ のとき, かつそのときに限り, B' で $([s]_a,[s]_b,[s]_1,...,[s]_n) \xrightarrow{a} ([s']_a,[s']_b,[s']_1,...,[s']_n)$.

(証明) この証明は, 補題3において補題2のかわりに補題5を使ったものと同じである. □

定理 3 B をプロセスとし, $\{A_a,A_b\}$ を $Act(B)$ の分割とする. L_a,L_b, R_i $(i=1,2,...,n)$ をアルゴリズム3に B,A_a,A_b を入力して得られたプロセスとする. このとき,

$$B \sim (L_a \,|||\, L_b)\,|G|\,(R_1 \,|||\, ... \,|||\, R_n).$$

(証明) この証明は, 補題5と補題6を使って定理2の証明と同様に示せる. □

例 **6** 図 4.2 (a) に示すプロセス B を分割する.

$Act(B) = \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, x, y, z\}$ を $A_a = \{a_1, a_2, a_3, a_4, x\}$ と $A_b = \{b_1, b_2, b_3, y, z\}$ に分割すると仮定する.

アルゴリズム 3 は, 次のステップを経て B を分割する.

(Step 1): ここでは, 図 4.2 (b) に示すプロセスを生成する.

(Step 2): Proc (s, a) を次の 3 つの非決定的状態に適用する: $s_0 \xrightarrow{x} s_1 + s_2$, $[s_3]_b \xrightarrow{y} [s_6]_b + [s_{10}]_b$, $[s_6]_b \xrightarrow{z} [s_9]_b + [s_{13}]_b$. 最終的に図 4.2 (c) に示されるプロセスを構成する.

定理 3 より, $B \sim (L'_a \,|||\, L'_b) \,||\, (R'_1 \,|||\, R_3)$ が成立する.

4.3.3 段階 3

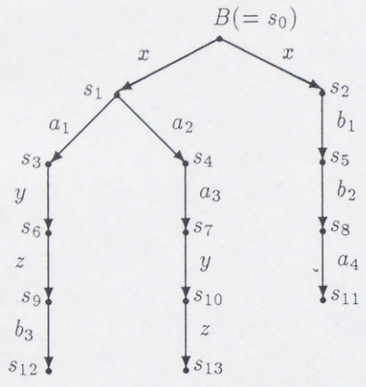
段階 2 までに残っていた制限は, 内部アクション **i** を入力仕様に記述しないというものである. 本段階では, この制限を取り除き, 一般のプロセスに対する分割法を与える.

内部アクション **i** は, 観測可能なアクションと異なり, 他のアクションとの同期なしで生起する. この性質を考慮すると, **i** を含むプロセスの分割として, 次のような戦略が考えられる.

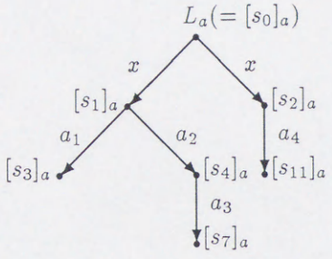
まず入力仕様 B からすべての **i** を取り除く. この操作は, **i** がラベル付けされた遷移の前状態と後状態を同じ状態とみなすことで実現できる. こうして得られたプロセスに対してアルゴリズム 3 を適用する. 結果として得られたプロセス L_a, L_b, R_i に対し, これらの並列合成が B と等価になるような適切な状態に **i** の遷移を挿入する.

以上の操作を例題を使って説明する.

例 **7** 図 4.3 (a) のプロセス B を分割する. 分割の基準として, $Act(B) = \{a_1, a_2, b_1, b_2, \}$ を $A_a = \{a_1, a_2\}$ と $A_b = \{b_1, b_2\}$ に分割すると仮定する.

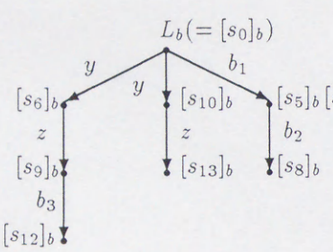


(a) 分割前のプロセス B

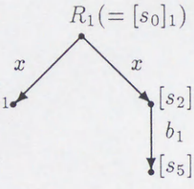


$[s_2]_a = \{s_2, s_5, s_8\}$
$[s_3]_a = \{s_3, s_6, s_9, s_{12}\}$
$[s_7]_a = \{s_7, s_{10}, s_{13}\}$

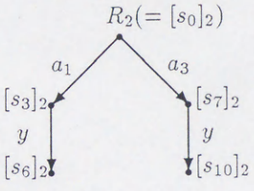
(b) 分割されたプロセス



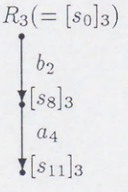
$[s_0]_b = \{s_0, s_1, s_2, s_3, s_4, s_7\}$
$[s_8]_b = \{s_8, s_{11}\}$



$[s_1]_1 = \{s_1, s_3, s_4, s_6, s_7, s_9, s_{10}, s_{12}, s_{13}\}$
$[s_5]_1 = \{s_5, s_8, s_{11}\}$

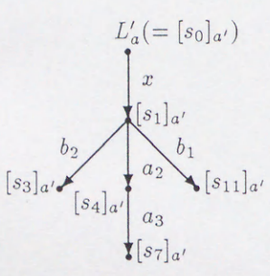


$[s_0]_2 = \{s_0, s_1, s_2, s_4, s_5, s_8, s_{11}\}$
$[s_6]_2 = \{s_6, s_9, s_{12}\}$
$[s_{10}]_2 = \{s_{10}, s_{13}\}$

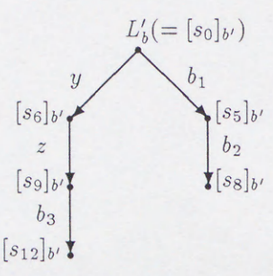


(Omitted)

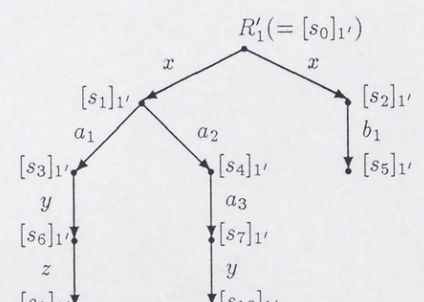
(b) (つづき)



$[s_1]_{a'} = \{s_1, s_2, s_5, s_8\}$
$[s_3]_{a'} = \{s_3, s_6, s_9, s_{12}\}$
$[s_7]_{a'} = \{s_7, s_{10}, s_{13}\}$



$[s_0]_{b'} = \{s_0, s_1, s_2, s_3, s_4, s_7\}$
$[s_6]_{b'} = \{s_6, s_{10}\}, [s_8]_{b'} = \{s_8, s_{11}\}$
$[s_9]_{b'} = \{s_9, s_{13}\}$



$[s_5]_{1'} = \{s_5, s_8, s_{11}\}$

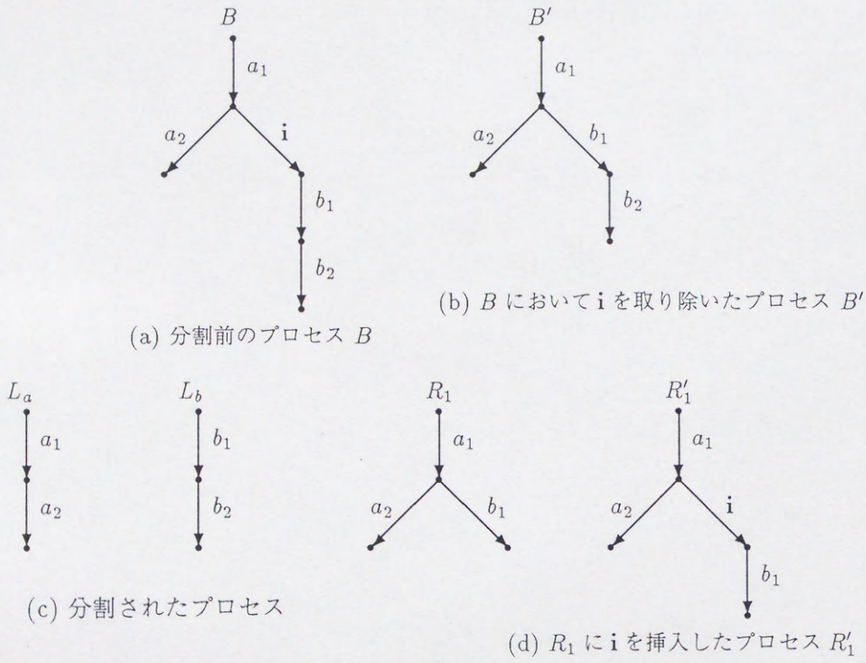
(c) アルゴリズム 3 からの出力

図 4.2: 例題 6 における LTSs

まず, B から $\mathbf{i}$ を除いた B' を準備する. これは図 4.3 (b) に示される. 次に, B', A_a, A_b に対してアルゴリズム 3 を適用すると図 4.3 (c) に示されるプロセスが得られる.

プロセス $(L_a \,|||\, L_b) \,|[a_1, a_2, b_1]|\, R_1$ は B' と強双模倣等価である. しかし元の B とは等価ではない.

この例の場合, B と強双模倣なプロセスを次のようにして得ることが可能である. すなわち, R_1 を R_1' に変更する (図 4.3 (d) 参照). この結果, $B \sim (L_a \,|||\, L_b) \,|[a_1, a_2, b_1]|\, R_1'$ となる.



(a) 分割前のプロセス B

(b) B において $\mathbf{i}$ を取り除いたプロセス B'

(c) 分割されたプロセス

(d) R_1 に $\mathbf{i}$ を挿入したプロセス R_1'

図 4.3: 例題 7 における LTSs

入力仕様中での $\mathbf{i}$ の位置を記録し, 分割後に元の位置へ挿入するために, 次の定義が必要である.

定義 20 $\mathcal{G}$ を, B の $\mathbf{i}$ とラベル付けされた枝だけから構成されたサブグラフとする (図 4.4 参照). このサブグラフ $\mathcal{G}$ に対し, $\mathcal{G}'$ を次のようにし

て B から構成されたサブグラフとする. すなわち $s \xrightarrow{a} s'$ を $\mathcal{G}$ に加えるのは, $a \neq \mathbf{i}$, $s \in \mathcal{G}$ あるいは $s' \in \mathcal{G}$ のとき, かつそのときに限る. この結果導出された $\mathcal{G}'$ をセル (cell) と呼ぶ.

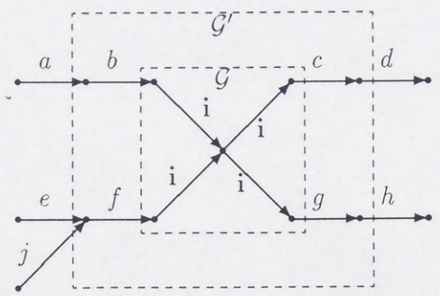


図 4.4: $\mathcal{G}$ と $\mathcal{G}'$ の例 (状態名は省略).

本章で構成するアルゴリズムは, セルに関してパターンマッチングを用いている. まず, パターンマッチングテーブル $\mathcal{T}$ を準備する. $\mathcal{T}$ は, すべての $(\mathcal{G}', \mathcal{G}'')$ の集合から構成される. ここで, $\mathcal{G}'$ は B 中のセルであり, $\mathcal{G}''$ は $\mathcal{G}'$ からすべての $\mathbf{i}$ を取り除いたサブグラフである.

アルゴリズム 4

入力: B, A_a, A_b.

出力: L_a, L_b, ここで $Act(L_a) = A_a, Act(L_b) = A_b$. $R_i, (i = 1, 2, ..., n)$, ここで各 $Act(R_i)$ は互いに素である.

(Step 1) B のパターンマッチングテーブル $\mathcal{T}$ を構成する.

(Step 2) B と $Act(B)$ に副アルゴリズム 2 を適用し, B から $\mathbf{i}$ とラベル付けられた枝を取り除く. この結果 B' を得る.

(Step 3) B', A_a, A_b にアルゴリズム 3 を適用し, L_a, L_b, R_i を得る.

(Step 4) 以下のように, $\mathcal{T}$ に基づいて $\mathbf{i}$ を L_a, L_b, R_i の中の対応する位置に挿入する:

1. 各 R_i $(i = 1, 2, ..., n)$ に対し，次の規則を繰り返し適用する．もし R_i 中のあるサブグラフが，マークされていない $(\mathcal{G}', \mathcal{G}'') \in \mathcal{T}$ の $\mathcal{G}''$ にマッチするならば，このサブグラブを $\mathcal{G}'$ で置き換える．そして $(\mathcal{G}', \mathcal{G}'')$ をマークする．

2. 各 L_α $(\alpha = a, b)$ に対し，次の規則を繰り返し適用する．もし L_α 中のあるサブグラフが，マークされていない $(\mathcal{G}', \mathcal{G}'') \in \mathcal{T}$ の $\mathcal{G}''$ にマッチするならば，このサブグラブを $\mathcal{G}'$ で置き換える．そして $(\mathcal{G}', \mathcal{G}'')$ をマークする．

ここでアルゴリズム 4 の適用例を示す．

例 8 図 4.5 (a) のプロセス B を分割する．$Act(B) = \{a, b_1, b_2, b_3, b_4\}$ を $A_a = \{a\}$ と $A_b = \{b_1, b_2, b_3, b_4\}$ に分割すると仮定する．

アルゴリズム 4 の適用過程は次のとおりである．

(Step 1) $\{(\mathcal{G}'_1, \mathcal{G}''_1), (\mathcal{G}'_2, \mathcal{G}''_2)\}$ を作る (図 4.5 (b))．

(Step 2) B' を構成する (図 4.5 (c))．

(Step 3) アルゴリズム 3 を B' に適用し，L_a, L_b, R_1 を構成する (図 4.5 (d))．

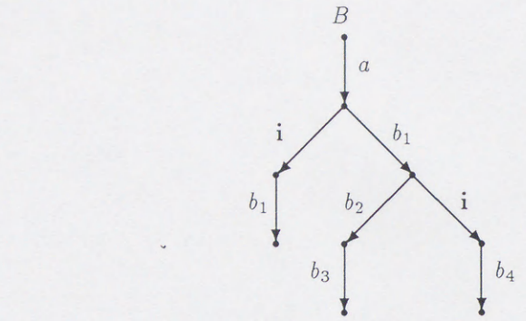
(Step 4) R_1 における $\mathcal{G}''_1$ を $\mathcal{G}'_1$ で置き換える．次に，$\mathcal{G}''_2$ については L_b と in R_1 の両方に存在するが，アルゴリズム 4 では，R_1 における $\mathcal{G}''_2$ を $\mathcal{G}'_2$ に置き換える (図 4.5 (e))．

最終的に得られたプロセスの並列合成 $(L_a \ ||| \ L_b) \ |[a, b_1, b_2, b_4]| \ R'_1$ は，B と強双模倣等価である．

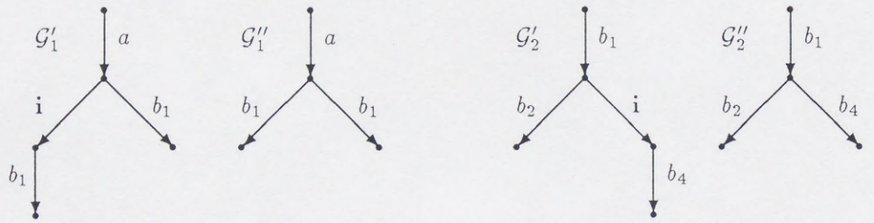
アルゴリズム 4 の性質は次のとおりである．

定理 4 B をプロセスとし，$\{A_a, A_b\}$ を $Act(B)$ の分割とする．L_a, L_b, $R_i (i = 1, 2, ..., n)$ をアルゴリズム 4 に B, A_a, A_b を入力して得られたプロセスとする．このとき，

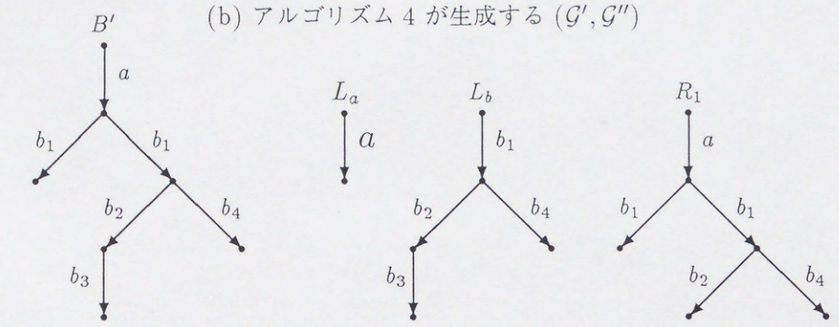
$$B \sim (L_a \ ||| \ L_b) \ |G| \ (R_1 \ ||| \ ... \ ||| \ R_n).$$



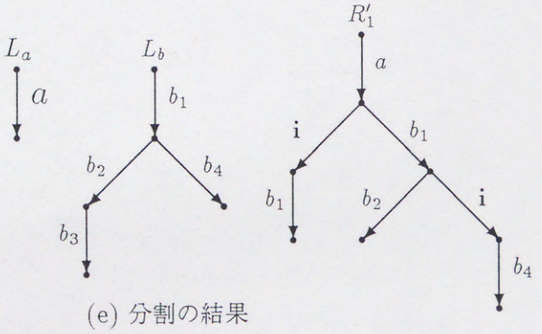
(a) 分割前のプロセス B



(b) アルゴリズム 4 が生成する $(\mathcal{G}', \mathcal{G}'')$



(c) プロセス B' (d) 分割されたプロセス



(e) 分割の結果

図 4.5: 例題 8 の LTSs とアルゴリズム 4 が生成する $(\mathcal{G}', \mathcal{G}'')$ (状態名は省略).

(証明) アルゴリズム 4 の手続きと, 内部アクションの性質から明らかである. □

4.4 適用例

本節では仕様の分割例を示す. 分割対象の仕様として, OSI 参照モデルにおける応用層サービスの一つのアソシエーション制御サービスを用いる [14].

本節では, この標準文書が規定しているサービスの動作を LTS として記述したものを分割する. これは図 4.6 に示される. ここで, ACS を構成するアクション集合は次のとおりである.

$$Act(ACS) = \{AReq1, AInd2, ARsp2a, ARsp2r, ACnfr, ACnf1a, ACnf1r, RReq1, RInd2, RRsp2a, RRsp2n, RCnf1a, RCnf1n\}.$$

分割の基準として, $Act(ACS)$ を次のように分割する.

$$A_a = \{AReq1, ACnfr, ACnf1a, ACnf1r, RReq1, RCnf1a, RCnf1n\},$$
$$A_b = \{AInd2, ARsp2r, ARsp2a, RInd2, RRsp2a, RRsp2n\}.$$

ここで, A_a は User-a 側で生起するアクションの集合であり, A_b は User-b 側で生起するアクションの集合である.

いま, プロセス ACS と 2 つのアクション集合 A_a, A_b をアルゴリズム 4 に入力すると, 図 4.7 と図 4.8 の 2 つの L と, 図 4.9 の 6 つの R が得られる.

定理 4 により, これらを並列合成したプロセス $(L_1 \;|||\; L_2) \;||\; (R_1 \;|||\; R_2 \;|||\; ... \;|||\; R_6)$ は ACS と強双模倣等価である.

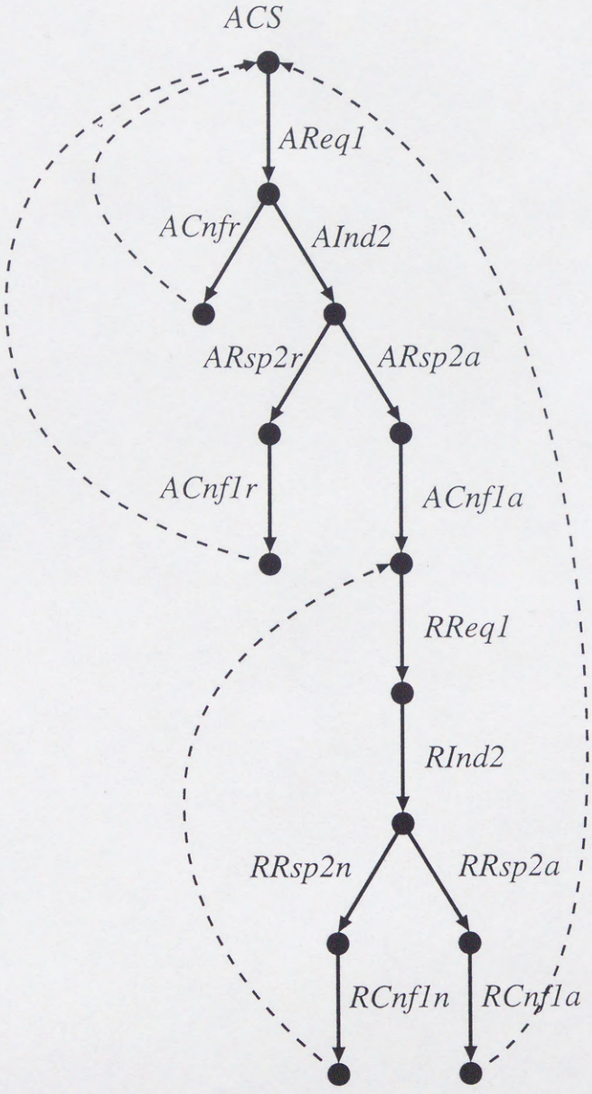


図 4.6: アソシエーション制御サービスの LTS. (点線で結ばれた状態は同一状態を意味する; 状態名は省略).

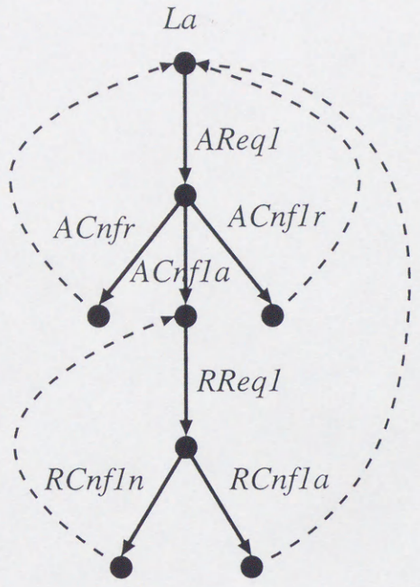


図 4.7: アソシエーション制御サービス分割の結果: L_a. (点線で結ばれた状態は同一状態を意味する; 状態名は省略).

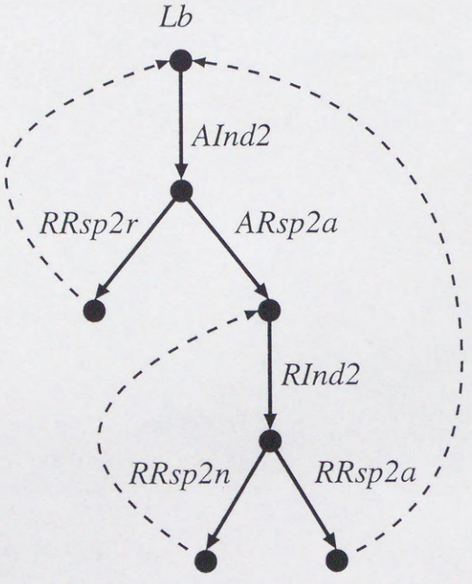


図 4.8: アソシエーション制御サービス分割の結果: L_b. (点線で結ばれた状態は同一状態を意味する; 状態名は省略).

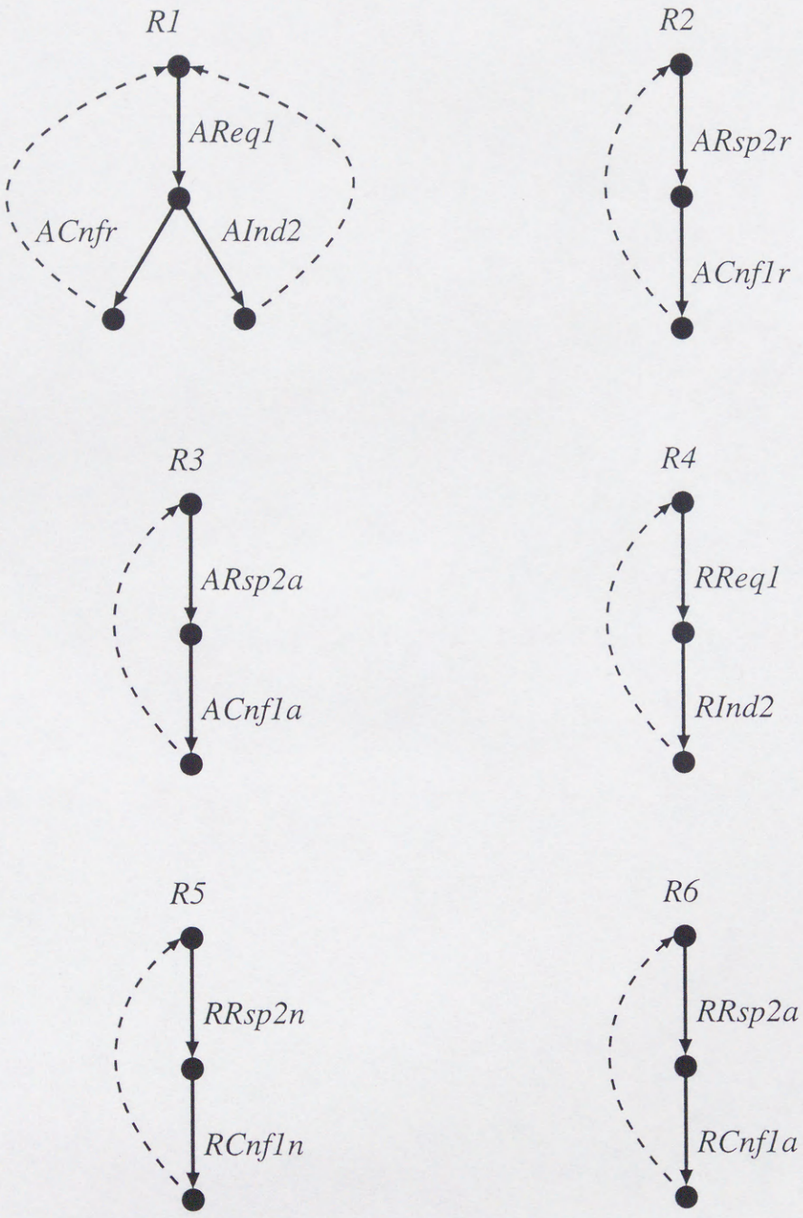


図 4.9: アソシエーション制御サービス分割の結果: R. (点線で結ばれた状態は同一状態を意味する; 状態名は省略).

4.5 まとめ

本章では次の2点を目的に研究を行った.

1) 第3章で構成した分割法を一般化し, 任意のプロセス代数仕様に適用できるようにする.

2) 第3章で構成した分割法で分割可能な仕様の領域を広げる.

以上の目的を達成するため, 本章では次のような研究を行った.

まず 1) について, 一般のプロセス代数仕様の意味表現である LTS に対して, 等価性を保つ分割法を構成することで, これを実現した.

次に 2) について, アルゴリズムの構成を3段階に分け, 段階的に制限を減らすことで, 最終的に分割可能なプロセスに対する制限を取り除いた.

以上の結果により, 前章での結果に比べ分割法がより実用的となっている. さらに, 各段階で構成したアルゴリズムに対し, 分割前のプロセスと分割後のプロセスとが等価性を保存することを証明しているため, 本分割法は, 高信頼なシステムを設計する上で特に有効である.

第 5 章

ソフトウェアのやわらかい開発法

5.1 はじめに

情報通信システム利用者の急激な増加にともない, これらに対する利用者要求は多種多様化し, さらに刻々と変化している. このような状況に柔軟に対処する新しい情報処理パラダイムとして, 近年「やわらかいシステム」あるいは「やわらかい情報処理」という概念が注目されている [15, 16, 17].

文献 [16] ではやわらかいシステムを 1) 知性, 2) 恒常性, 3) 適応性の 3 性質を持ったシステムと定義している. 本章では, 3) の適応性を実現するための基盤技術として, やわらかい開発法を議論する.

本研究が目指しているやわらかい開発法の実現イメージは, 次のとおりである. すなわち, 利用者要求や, システムが提供するサービスに変化が生じた場合, これらの変化を開発支援システムが自動的に吸収し, 新しい利用者要求や, システムの新しいサービスを効率的に実現する. この実現のために, 本章では変更要求の実装を効果的に支援するやわらかい開発法について考察する. まずやわらかいシステムとの位置付けを明確化し, その枠組を構成する. さらに仕様記述の観点から, その具体化について議論する.

本研究において FDTs をもとにやわらかい開発法を構成する理由は,

次の観点に基づいている． FDTs を用いたシステム設計では，システムの仕様に対する変更項目の定量的評価が期待でき [18]，かつシステムの機械的検証が可能になるため，システムの変更を系統的に取り扱うことができる．しかしながら，形式仕様上での変更を扱った研究は少ない [19]． 形式仕様上で，仕様の変更要求に対し，既存のシステム仕様を再利用しこれに変更を加えて新しいシステム仕様を構成できるのが望ましい．

本章の構成は次のとおりである．第 5.2 節でやわらかい開発法の枠組を示す．次に第 5.3 節でやわらかい開発法を具体的に議論する．第 5.4 節は本章のまとめである．

5.2 やわらかい開発法

5.2.1 やわらかいシステム

文献 [16] では，やわらかいシステムを次のように定義している．

定義 21 (やわらかいシステム) やわらかいシステムは，以下の特性を持つ情報システムである．

1) 知性: 環境を認識し，適切な応答を行なう能力であり，利用者要求の柔軟な認識と実時間での実現を行なう．

2) 恒常性: 一過性の環境の変化や自己組織の変動に対して安定的に機能を維持する．

3) 適応性: 長期的な環境の変化に対して，自己の組織の変更を行ない，機能の追加・修正を行なうことによる環境適応性を持つ．

恒常性と適応性は互いに競合する性質であり，この良い均衡を持つ情報システムは理想的なやわらかい情報システムである．

このやわらかいシステムに対する位置付けを明確化するため, まず本章で用いるやわらかいシステムの定義を示し, 次にやわらかい開発法を定義する.

ここでは要求とそれを満足するシステムの観点から, やわらかいシステムを定義する. 以下, システム S が要求 R を充足することを S **sat** R と表記し, 充足しないことを S $\not$**sat** R と表記する.

定義 22 (やわらかいシステム) R を要求, S をシステムとし, S **sat** R とする. さらに R' を R に変更を加えた要求とする.

もし S **sat** R' ならば, S は R と R' に対してやわらかいシステム という.

定義 22 は図 5.1 のように示される.

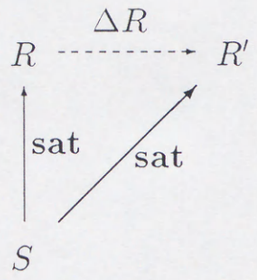


図 5.1: R と R' に対してやわらかいシステム S; ここで R, R' は要求を表わし, ΔR は R と R' の差分を表わす.

定義 22 のやわらかいシステムは, 定義 21 のやわらかいシステムの枠組に含まれる狭義の定義である. 以下本稿では, 定義 21 によるシステムの持つやわらかさを広義のやわらかさと呼び, 定義 22 によるシステムの持つやわらかさを狭義のやわらかさと呼ぶ.

5.2.2 やわらかい開発法の定義

本章では, やわらかい開発法を次の観点から定義する. システムが狭義のやわらかさを持ち, 利用者の新しい要求を満足する場合には, システムを再構築する必要はない. このことから, システムの再構築が必要となるのは, 狭義のやわらかいシステムでは充足不可能な要求がだされた場合である. すなわち, やわらかい開発法は次のように示される.

定義 23 (やわらかい開発法) R を要求, S をシステムとし, S **sat** R とする. さらに R' を R に変更を加えた要求とし, S $\not$**sat** R' とする.

このとき, R, R' と S をもとに S' **sat** R' となるようなシステム S' を構成する方法を, やわらかい開発法という.

定義 23 は図 5.2 のように示される.

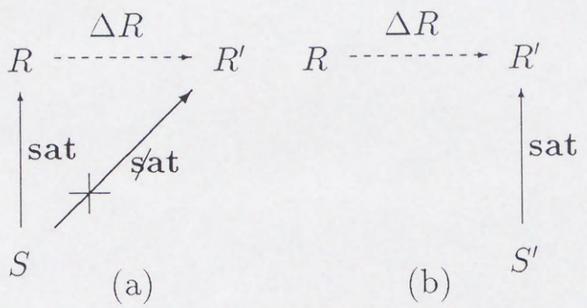


図 5.2: やわらかい開発法. (a) やわらかい開発法が必要になる状況; S は, 要求 R と R' に対してやわらくないシステム; ΔR は R と R' の差分を表わす. (b) やわらかい開発法により構成されたシステム S' の持つべき性質.

5.3 やわらかい開発法の具体化

本節では, 前節で定義したやわらかい開発法を具体化するにあたり, FDT 上でやわらかい開発法を実現する. ここに FDT を導入することで要求の差を明確にすることができ, その差に応じた系統的な支援が期待できるからである.

まず, 通信システムを LTS でモデル化する. そして, LTS で表現された通信システムのサービス仕様における変更要求を, 複数の LTS の組で表現されたプロトコル仕様に反映させるという課題に取り組む.

図 5.3 に本研究で対象としているシナリオを示す. 既存のサービス仕様と, あるプロトコル合成法によって合成されたプロトコル仕様を考える. これらの間には弱双模倣関係が成立しているとする. いまサービスに対して新たな要求が加えられ, サービス仕様が修正されたとする. 本研究の目的は, この修正後のサービス仕様から再度プロトコル仕様を合成することなく, 既存のプロトコル仕様を修正することで, 修正後のサービス仕様と弱双模倣等価なプロトコル仕様を構成することである.

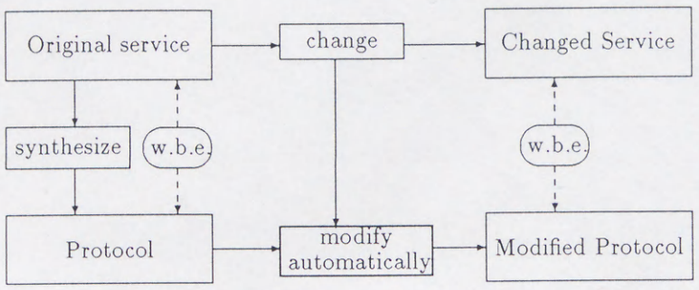


図 5.3: 本研究のシナリオ

本節では, 既存のサービス仕様への変更要求を 2 つのクラスに分ける. すなわち, 変更が並行プロセスとして表現できるクラスとそれ以外のクラスである. 前者においては, この変更をサービス仕様で並行プロセスとして実現する方法を示す. 次に, それぞれのクラスに対して, サービス仕様の変更をプロトコル仕様に反映させる手法を提案する.

5.3.1 準備

まず, 本章で提案する変更法の準備として, 以下の 3 項目を定義する. 1) LTS 上の演算, 2) サービスとプロトコルのモデル, 3) サービスから

のプロトコル合成法.

1) LTS に対する演算

ここでは, LTS におけるイベントの隠蔽を定義する. 隠蔽オペレータは, 指定したイベントを外部から観測不可能なイベントに置き換えることによって, そのイベントをシステム内部に隠すために使われる.

定義 24 (LTS に対するイベントの隠蔽) LTS $TS = (S, L, T, s_0)$ のイベント集合 A に関する隠蔽 $TS\backslash A$ を次の LTS で定義する.

$$(S, L - A \cup \{\mathbf{i}\}, T_h, s_0)$$

ここで $T_h \subseteq S \times [L - A \cup \{\mathbf{i}\}] \times S$ は次を満たす要素で構成された最小の集合で定義される.

もし $(s, e, s') \in T$ かつ $e \notin A$ ならば $(s, e, s') \in T_h$,
もし $(s, e, s') \in T$ かつ $e \in A$ ならば $(s, \mathbf{i}, s') \in T_h$.

なお, 本章では表現を簡潔にするため, 2 つの LTSs の並列合成 $TS_1 \,|A|\, TS_2$ を $TS_1 \,|_A\, TS_2$ と記述する.

2) サービスとプロトコルのモデル

本稿ではサービス仕様を, ラベル集合 L_0 を持った 1 つの LTS TS_0 で定義し, プロトコル仕様を次の LTS で定義する.

$$(TS_1 \,|_M\, TS_2)\backslash M.$$

ここで TS_1 と TS_2 はそれぞれラベル集合 L_1, L_2 を持った LTSs であり, プロトコルエンティティの仕様を表わす. M は同期イベントの集合でありエンティティが通信するためのチャネルを表わす (図 5.4 参照).

3) サービスからのプロトコル合成法

定義 25 (サービス仕様からのプロトコル仕様の合成問題) サービス仕様 TS_0 とイベント集合 A_0 (ただし $Act(TS_0) \subseteq A_0$) の分割 $\{A_1, A_2\}$ を

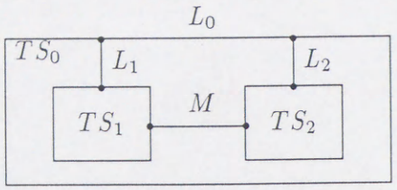


図 5.4: サービスとプロトコルのモデル

もとに, 次の条件を満たすプロトコルエンティティの仕様 TS_1, TS_2 を合成する.

$$TS_0 \approx (TS_1 \mid_M TS_2) \backslash M.$$

ただし $Act(TS_1) \cap A_0 = A_1, Act(TS_2) \cap A_0 = A_2$ である. M は $M \cap A_0 = \emptyset$ という性質を持つ同期イベントの集合であり, エンティティどうしが通信するための通信媒体を表わす.

議論を簡単にするため, 本章ではサービス仕様 $TS_0 = (S_0, L_0, T_0, s_{00})$ は次の条件を満足するものとする. 1) $\mathbf{i} \notin L_0$, 2) 任意の $s \in S_0$ に対し, $Init(s) \subseteq A_1$ あるいは $Init(s) \subseteq A_2$ のいずれかである.

ここで挙げた条件は議論を簡単化するためのものであり, 本質的な制限ではない. なぜならば, 1) については隠蔽オペレータを使って $\mathbf{i}$ を表現でき, 2) についてはアクションが公平に選択されるという仮定のもと, ポーリング機構 (polling mechanism) [13] を使うことで条件を除くことができるからである.

次に, サービス仕様とそれを構成するイベント集合の分割から, サービス仕様と等価なプロトコル仕様を導出するための方法を示す. ここでの方針として, サービス仕様の 1 つの状態に対し, 2 つのプロトコルエンティティにそれぞれ対応する状態を作ることにする. その上で次の 2 つのステップを実行する.

(1) サービス仕様の各遷移に対応するプロトコルの遷移を構成する.

(2) サービス仕様の遷移の系列がループしている場合に，プロトコル仕様においても対応するループを構成する．

(1) において，サービスの各遷移に対する実際の処理は次のとおりである．サービス仕様とそれから合成されたプロトコル仕様が弱双模倣等価になるためには，プロトコルエンティティどうしが同期をとりあい，サービス仕様の動作と同じ動作をするようにしなければならない．サービス仕様におけるあるイベントに対応するイベントが，あるエンティティで生起した場合，このイベントが同期情報を別のエンティティに伝えるべきかどうかは，サービスにおけるそのイベントの直後に生起可能なイベントに依存する．

(1-a) サービス仕様においてある遷移をとったとき，もし直後の遷移が同じイベント集合の要素であれば，その遷移はプロトコルエンティティにおいて局所化されている．そのためその遷移による状態遷移を別のエンティティに伝える必要はない．このとき別のエンティティでは，そのイベントの生起によって状態が遷移しないため，サービス仕様におけるその遷移に関連した2つの状態は，同一状態とみなせる．これを表現するために，アルゴリズムでは状態集合を使っている．結果として得られるLTSでは，状態集合が通常のLTSの状態に相当している．

(1-b) サービス仕様においてある遷移をとったとき，もし直後の遷移が別のイベント集合の要素であれば，プロトコルエンティティにおいてその遷移に対応する遷移があったことを別のエンティティに同期情報として伝えることが必要である．したがってその遷移の後に同期イベントを持った遷移を加え，これと同期するイベントを持った遷移を別のエンティティに加える．

(2) は，(1) で構成された共通要素を持つ状態集合の集合に対し，共通要素を持つ2つの集合の和集合を，これらに置き換えることで実現される．結果として得られる状態集合は，互いに素な要素を持つ集合となる．

この処理は前章で構成した手続き $PWDJ$ によって実現される.

以下に, プロトコルを合成するアルゴリズムと, その中で使われている手続きを示す.

アルゴリズム 5:

入力: $TS_0 = (S_0, L_0, T_0, s_{00})$, A_1, A_2.

出力: $TS_1 = (\mathcal{S}_1, L_1, T_1, s_{10})$, $TS_2 = (\mathcal{S}_2, L_2, T_2, s_{20})$.

準備: $\mathcal{S}_1 \leftarrow \{\{s_{10}\}\}, T_1 \leftarrow \emptyset, L_1 \leftarrow A_1, \mathcal{S}_2 \leftarrow \{\{s_{20}\}\}, T_2 \leftarrow \emptyset, L_2 \leftarrow A_2$.

{ (1) において $m \in \{1,2\}, n \in \{1,2\}, m \neq n$ とする. }

{ (1-b) において, 状態 s に対し s' は新しいユニークな状態, }

{ e に対し e' は新しいユニークな (fresh) イベントを意味する. }

begin

foreach $(s_{0i}, e, s_{0j}) \in T_0$ **do** (1)

begin

if $e \in A_m$ *and* $Init(s_{0j}) \subseteq A_m$ **then** (1-a)

begin

$\mathcal{S}_m \leftarrow \mathcal{S}_m \cup \{\{s_{mi}\}, \{s_{mj}\}\}$; $T_m \leftarrow T_m \cup \{(s_{mi}, e, s_{mj})\}$;

$\mathcal{S}_n \leftarrow \mathcal{S}_n \cup \{\{s_{ni}, s_{nj}\}\}$

end;

if $e \in A_m$ *and* $Init(s_{0j}) \neq \emptyset$ *and* $Init(s_{0j}) \subseteq A_n$ **then** (1-b)

begin

$\mathcal{S}_m \leftarrow \mathcal{S}_m \cup \{\{s_{mi}\}, \{s'_{mi}\}, \{s_{mj}\}\}$;

$T_m \leftarrow T_m \cup \{(s_{mi}, e, s'_{mi}), (s'_{mi}, e', s_{mj})\}$; $L_m \leftarrow L_m \cup \{e'\}$;

$\mathcal{S}_n \leftarrow \mathcal{S}_n \cup \{\{s_{ni}\}, \{s_{nj}\}\}$; $T_n \leftarrow T_n \cup \{(s_{ni}, e', s_{nj})\}$;

$L_n \leftarrow L_n \cup \{e'\}$

end

end;

$\mathcal{S}_1 \leftarrow PWDJ(\mathcal{S}_1)$;　$\mathcal{S}_2 \leftarrow PWDJ(\mathcal{S}_2)$　(2)

end.

アルゴリズム 5 によって合成された LTSs と, 元の LTS には次の性質が成立する.

定理 5 サービス仕様 TS_0 とイベント集合 A_0 (ただし $Act(TS_0) \subseteq A_0$) の分割 $\{A_1, A_2\}$ をもとに, アルゴリズム 5 によって構成されたプロトコルエンティティの仕様 TS_1, TS_2 との間には $TS_0 \approx (TS_1 \mid_M TS_2) \backslash M$ という関係が成立する. ここで $M = Act(TS_1) \cap Act(TS_2)$. □

5.3.2 サービス仕様への変更要求に対するプロトコル仕様の変更

サービス仕様への変更要求を分類する. 最も基本的は変更要求は次の二つに分けられる.

1. 追加: 仕様に情報を加えること, LTS では遷移関係の要素を増やすことに相当する.
2. 削除: 仕様から情報を一部取り除くこと, LTS では遷移関係の要素を減らすことに相当する.

次の 2 つの理由から, 仕様の変更部分が既存の仕様から独立したコンポーネントであるのが望ましい.

1. 新しいコンポーネントを単独で論理検証することができる.
2. 既存の仕様が実装されていた場合に, 変更部分だけを独立に実装できる.

本章では, 並列合成オペレータを使い, まずサービス仕様において変更目標を並行プロセスとして表現する手法を与え, 次にこの並行プロセス

相当する変更をプロトコル仕様において実現する手法を与える．ただし，この手法は任意のサービス仕様とその変更に対して適用できるものではない．そこでまず仕様とその変更を，この手法を適用できるクラスとできないクラスに分ける．

追加

追加要求は 1) 並列合成オペレータを使って実現可能なクラスと 2) それ以外のクラスに分けられる．既存の仕様に次の 2 つの条件が成立するものをクラス 1) とする．

C1：既存の仕様にないイベントを追加する．

C2：このイベントによってラベル付けされた遷移を追加する場所 (状態) が前後のイベントによって一意に決定できる．

既存の仕様において，変更の対象となる状態を前後のイベントによって一意に定めることができる場合には，元の仕様を変更せずに変更目標を，並行 LTS を一つ加えるだけで実現することが可能である．これは並列オペレータの定義からわかる．

並列合成オペレータを使って実現可能なクラス　典型的な追加要求は，既存の LTS にない新しいイベントを導入し，これを LTS に追加することである．

この節でのシナリオを図 5.5 に示す．TS_0 は既存のサービス仕様であり，$(TS_0 \mid_M TS_2)\backslash M$ はそのサービス仕様から合成されたプロトコル仕様である．これらは弱双模倣等価である．ここでは，サービス仕様の変化を新しい並行プロセスを並列合成オペレータでつなぐことで表現する．すなわち変化後のサービス仕様は $TS_0 \mid_{C_0} TS_0'$ である．本章では，この変化分 TS_0' に相当するプロトコル仕様を合成し，既存のプロト

コル仕様を再利用して, 変化後のサービス仕様と弱双模倣等価なプロトコル仕様を構成する. これは $((TS_1 \mid_{C_1} TS_1') \mid_{M'} (TS_2 \mid_{C_2} TS_2'))\backslash M'$ と表現される.

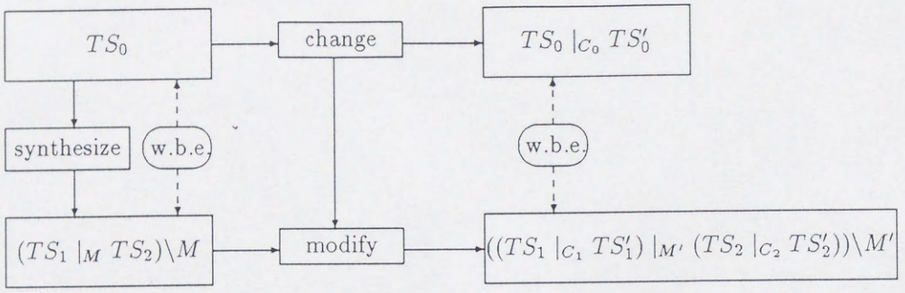


図 5.5: シナリオ

目標仕様　既存のサービス仕様にない新しいイベントを導入し, サービス仕様中に差し挟む. 図 5.6 に対して, 次の 3 つが考えられる:

1. 状態分割 (separation) と遷移の追加 (図 5.7).
2. 自己ループの追加 (図 5.8).
3. 分岐による新しいシステムの追加 (図 5.9).

サービス仕様において, 変更の対象となる状態を目標状態という. 図 5.6 では, s_j が目標状態である.

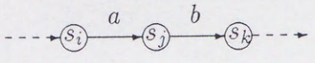


図 5.6: 元となる LTS の一部

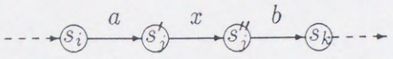


図 5.7: 状態分割と遷移の追加

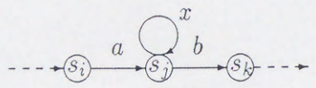


図 5.8: 自己ループの追加

サービス仕様における目標仕様の実現手法　前述の目標を並行 LTS を使って実現する方法は次のとおりである.

サービス仕様を $TS_0 = (S_0, L_0, T_0, s_{00})$, 目標状態を $s_{obj} \in S_0$ とする.

(Step 1) 目標状態の前後の遷移から構成された LTS を作る. これは, 第 4 章で構成した Rest を使って次のように実現できる.

$$TS_0' \leftarrow Rest(TS_0, Pre(s_{obj}) \cup Init(s_{obj}))$$

(Step 2) TS_0' において, 目標状態に相当する状態 s_{obj} に関する遷移を追加し, これを改めて TS_0' とする. 具体的には, 図 5.7, 5.8, 5.9 に従う.

この結果, $TS_0 \mid_{C_0} TS_0'$ が目標仕様となる. ここで $C_0 = Act(TS_0) \cap Act(TS_0')$ である.

並列オペレータの定義から以下の性質は明らかである.

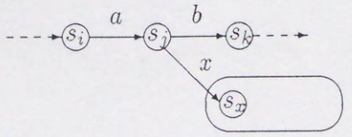


図 5.9: 分岐による新しいシステムの追加

性質 3 制限 C1 と C2 のもとで目標仕様は上記の手法を使って実現できる. □

プロトコル仕様の変更法 前節で挙げたサービス仕様への変更要求を, プロトコル仕様へ反映させる方法は以下のとおりである.

既存のサービス仕様と, それから合成されたプロトコル仕様を それぞれ $TS_0, (TS_1 \mid_M TS_2)\backslash M$ とする. さらに, 上記の手法で構成された目標仕様を $TS_0 \mid_{C_0} TS_0'$ とする.

TS_0' とイベント集合 A_0' (ただし $Act(TS_0') \subseteq A_0'$) の分割 $\{A_1', A_2'\}$ をアルゴリズム 1 に与え, LTSs TS_1', TS_2' を得る.

この結果, $(TS_1 \mid_{C_1} TS_1') \mid_{M'} (TS_2 \mid_{C_2} TS_2')$ がプロトコル仕様となり得る. ここで $C_1 = Act(TS_1) \cap Act(TS_1'), C_2 = Act(TS_2) \cap Act(TS_2'), M' = M \cup (Act(TS_1') \cap Act(TS_2'))$ である.

以上の手法から, 次の性質が成り立つ.

性質 4 サービス仕様 TS_0 とイベント集合 A_0 (ただし $Act(TS_0) \subseteq A_0$) の分割 $\{A_1, A_2\}$ をもとに, アルゴリズム 5 によって構成されたプロトコルエンティティの仕様 TS_1, TS_2 を仮定する.

いま変更要求の実現として TS_0' を TS_0 に並列合成するとする. すなわち, $TS_0 \mid_{C_0} TS_0'$. ここで, TS_0' とイベント集合 A_0' (ただし $Act(TS_0') \subseteq A_0'$) の分割 $\{A_1', A_2'\}$ をもとに, アルゴリズム 5 によって構成されたプロトコルエンティティの仕様 TS_1', TS_2' を仮定する.

このとき, $TS_0 \mid_{C_0} TS_0' \approx ((TS_1 \mid_{C_1} TS_1') \mid_{M'} (TS_2 \mid_{C_2} TS_2'))\backslash M'$ という関係が成立する. ここで $M' = M \cup (Act(TS_1') \cap Act(TS_2'))$. □

並列合成オペレータを使って実現できないクラス サービス仕様において目標状態がその前後のアクションによって一意に決定できない場合や, 複数の状態を変更する場合には, 前節で示した並行 LTS による目標仕様の

表現ができない．この場合，直接サービス仕様のLTSを書換えることが必要である．サービス仕様の変更に対するプロトコル仕様の変更法は，既存のサービス仕様におけるアクション集合の分割と変更部分に関して，次の2つに分けられる．

1. 局所的な場合．このとき，サービスの変更は1つのプロトコルエンティティにしか影響を与えない．具体的には次の性質が成立するものである．すなわち，サービス仕様において変更する状態への入出力遷移のイベントがアクション集合の分割の同じ要素にあり，さらに追加する遷移のイベントもすべてこの集合の要素にある．

2. 大域的な場合．これは局所的でない場合である．このとき，サービスの変更は2つのプロトコルエンティティに影響を与えてしまう．

このクラスのサービス仕様の変更に対して，プロトコル仕様を自動変更することが可能である．これは，本稿で用いた合成アルゴリズムがサービス仕様における各状態をプロトコル仕様の状態集合として持つように仕様を構成しているからである．

ここでは前者の場合について，サービス仕様とプロトコル仕様の変更手順を示す．既存のサービス仕様とプロトコル仕様をそれぞれ TS_0, $(TS_1 \mid_M TS_2)\backslash M$ とする．

(Step 1) サービス仕様 TS_0 を変更する．この時変更する状態を s_i, s_j とし，追加あるいは修正される遷移を (s_i, e, s_j) とする．

(Step 2) プロトコル仕様 $(TS_1 \mid_M TS_2)\backslash M$ において，イベント e を持つエンティティの仕様を TS_k とする．TS_k における状態 $[s_i], [s_j]$ に対し，(s_i, e, s_j) をサービス仕様において行ったものと同じ方法で追加あるいは修正する．

性質 5 上記の手順で変更したサービス仕様と修正したプロトコル仕様は弱双模倣等価である．□

削除

削除要求は, 1) 隠蔽オペレータを使って実現可能なクラスと 2) それ以外のクラスに分けられる.

1) のクラスの実現法は明らかである. これは, 隠蔽オペレータが特定の観測可能なイベントを隠蔽し, 外部から観測不能な内部イベントにすることを目的に作られているからである.

隠蔽オペレータを使って実現可能なクラス　既存のサービス仕様と, それから合成されたプロトコル仕様を それぞれ $TS_0, (TS_1 \mid_M TS_2)\backslash M$ とする. さらに, サービス仕様において削除したいイベントを a とする.

変更方法は次のとおりである. サービス仕様において $TS_0\backslash\{a\}$ とする. プロトコル仕様においても同様に, $((TS_1 \mid_M TS_2)\backslash M)\backslash\{a\}$ とすればよい.

性質 6 上記の手順で変更したサービス仕様と修正したプロトコル仕様は弱双模倣等価である. □

隠蔽オペレータを使って実現できないクラス　このクラスでは, 直接 LTS の一部を書換えることが必要になる. 第 5.3.2 節との類似で, ここでも削除が 2 つの場合に分けられる. (1) 局所的な場合, (2) 大域的な場合.

ここでは前者の場合について, サービス仕様とプロトコル仕様の変更手順を示す. 既存のサービス仕様とプロトコル仕様をそれぞれ TS_0, $(TS_1 \mid_M TS_2)\backslash M$ とする.

(Step 1) サービス仕様 TS_0 を変更する. この時変更する状態を s_i, s_j とし, 追加あるいは修正される遷移を (s_i, e, s_j) とする.

(Step 2) プロトコル仕様 $(TS_1 \mid_M TS_2)\backslash M$ において, イベント e を持つエンティティの仕様を TS_k とする. TS_k における状態 $[s_i], [s_j]$

に対し, (s_i, e, s_j) をサービス仕様において行ったものと同じ方法で追加あるいは修正する.

性質 7 上記の手順で変更したサービス仕様と修正したプロトコル仕様は弱双模倣等価である. □

5.3.3 適用例

ここでは, 本章で提案した, サービス仕様への変更要求に対するプロトコル仕様の自動変更法の例題を示す.

まずサービス仕様として, 図 5.10 の LTS をとる. これはメッセージ (message) を送信してから確認 (acknowledgement) を受けとるまでの一連のイベントを示したものである. これからアクション集合の分割 $\{\{sndmsg, recack\}, \{recmsg, sndack\}\}$ をもとにアルゴリズム 5 を適用すると, プロトコルエンティティの仕様, すなわち図 5.11 の LTSs が得られる. これらの間には次の関係が成立する.

$$TS_0 \approx (TS_1 \mid_M TS_2) \backslash M$$

ここで, $M = \{sndmsg', sndack'\}$ である.

いま, サービスに否定確認 (negative acknowledgement) に関するイベントシーケンスを加えるため, TS_0 の状態 s_{02} に分岐となる遷移を加える. この目標仕様は, 図 5.12 に示される. この目標状態 s_{02} は TS_0 において, その前後のアクションで区別できるできるため, この変更は並行 LTS を加えることで実現可能である. 図 5.13 に目標サービス仕様を実現するために 既存サービス仕様に加える並行 LTS を示す. 変更後の仕様は $TS_0 \mid_{C_0} TS_0'$ である. ただし $C_0 = \{recmsg, sndack\}$.

図 5.14 は, TS_0' と $\{\{recnak\}, \{recmsg, sndack, sndnak\}\}$ をアルゴリズム 5 に適用して得られたものである. これらの間には次の関係が成立する.

$$TS_0' \approx (TS_1' \mid_{M''} TS_2') \backslash M''$$

ここで, $M'' = \{sndnak'\}$ である.

最終的に図 5.14 の LTSs を 図 5.11 のプロトコル仕様に次のように加えることで, 変更後のサービス仕様 $TS_0 \mid_{C_0} TS'_0$ と弱双模倣等価なプロトコル仕様を得る:

$$((TS_1 \mid_{C_1} TS'_1) \mid_{M'} (TS_2 \mid_{C_2} TS'_2))\backslash M'.$$

ここで $C_1 = C_2 = C_0$, $M' = \{sndmsg', sndack', sndnak'\}$ である.

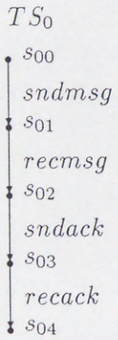


図 5.10: サービス仕様

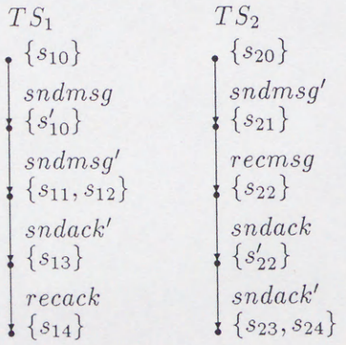


図 5.11: サービス仕様から合成されたプロトコルエンティティの仕様.

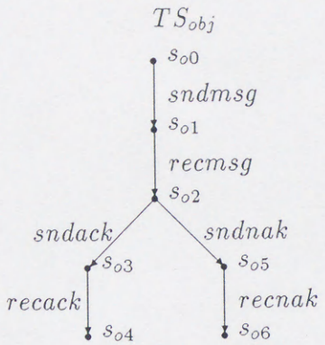


図 5.12: 目標仕様

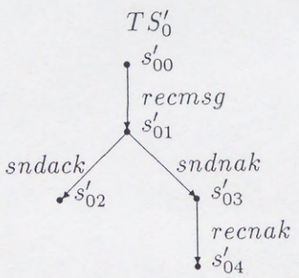


図 5.13: 目標仕様を表現するための並行 LTS.

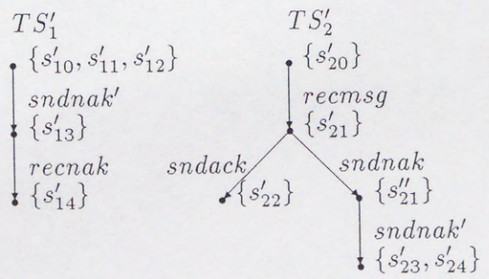


図 5.14: サービス仕様の変更部分から合成されたプロトコル仕様の一部.

5.4 まとめ

本章では段階的詳細化に適したやわらかい開発法を構成することを目的に, 次の 2 項目について議論した.

1) やわらかい開発法の概念的枠組.

2) 通信プロトコル合成に対する 1) の枠組の具体化.

1) では, やわらかい開発法について議論しその枠組を与えた. やわらかい開発法をやわらかいシステムを実現するための基盤技術として位置付け, その上でこれらの定義を行なった.

次に 2) では, 通信システムのサービス仕様における変更要求に対して, そのプロトコル仕様を自動的に変更する手法について述べた. 仕様の変更部分が既存のサービス仕様と独立なコンポーネントで表現されるのが望ましいという観点から, 目標の仕様を並行 LTS で実現する手法を構成し, それにもとづいてプロトコル仕様を自動変更する手法について述べた. さらに並行 LTS を使って実現できないクラスの変更についても考察し, その支援法を述べた.

本章の結果に対する今後の課題として, サービス仕様における変更の種類を増やすこと, 変更支援環境の構成が挙げられる.

第 6 章

結論

本論文では, 情報通信システムに代表される大規模・複雑なシステムの開発という問題に対し, 仕様の段階的詳細化というアプローチをとった. そのうえで, この仕様記述作業に有効な基盤技術として次の 2 つの点に焦点を当てた.

1) 系統的な仕様の分割法が必要である.

2) 開発工程の段階において, 前段階での仕様の変化を後段階へ効率的に伝える方法が必要である.

すなわち, 本研究では次の 2 点を目的に研究を行った.

a) 仕様の等価性を保存する分割法を構成する.

b) 前段階での仕様の変化に対する後段階の仕様の自動変更法を構成する.

a) の目的を達成するために, 第 3 章では, LOTOS 仕様の等価性にもとづく分割法を構成した. 次に, ここで構成した分割法を任意のプロセス代数仕様の分割法として一般化するために, 第 4 章では, LTS 仕様の分割法を構成した. これは, LTS が任意のプロセス代数仕様の意味表現として用いられているからである.

b) の目的を達成するために, 第5章では, ソフトウェアのやわらかい開発法の枠組を構成し, この実現として通信ソフトウェアのサービス仕様の変更に対するプロトコル仕様の変更法を構成した.

本論文で構成した手法は, すべて, 仕様の動的振舞いの等価性を保つことを目的にしている. そのため, 第3章と第4章の分割法については分割の前後の仕様が等価性を保つことを, 第5章の変更法については変更後のサービス仕様とプロトコル仕様が等価性を保つことを数学的に証明した. このことは, これらの手法が分割法・変更法として用いられる場合に, 望ましくない振舞いを追加することがないことを意味する.

以上, 本論文の結果は, 形式仕様の段階的詳細化を効果的に進めるための基盤として, 信頼できる技術を提供する. これらの手法を導入することで, 仕様の分割・変更作業にともなう仕様の検証作業の負担を軽減することができる. その結果, システム開発における生産性の大幅な向上を期待できる.

謝辞

本研究を行う機会を与えていただき, 熱心に御指導してくださいました, 東北大学電気通信研究所 白鳥則郎 教授に心から深く感謝いたします.

また研究内容について非常に多くの助言をいただきました, 東北大学電気通信研究所 富樫敦 助教授に深く感謝いたします.

本論文の審査をしていただきました東北大学工学部 根元義章 教授, 並びに, 東北大学大型計算機センター 牧野正三 教授に深謝いたします.

日頃からゼミで熱心な御討論と有益な助言をいただきました, 仙台電波工業高等専門学校 高橋薫 助教授に感謝いたします. また同校 加藤靖 教授, 安藤敏彦 助手からは研究に関して多くの助言をいただきました.

歴代の白鳥研のスタッフの方々には, たいへんお世話になりました. ここに名前を挙げて感謝いたします. 宮城教育大学 布川博士 助教授, 成均舘大学校李殷碩 助教授, 会津大学 Goutam Chakraborty 講師, 東北大学 Dusan Jokanovic 助手, 小野良司 助手.

秘書の谷垣訓子さん, 八巻美智子さんには事務関連でお世話になりました. ありがとうございました.

日頃から熱心に討論していただいた東北大学白鳥研究室, 並びに, 旧野口研究室の皆様に深く感謝いたします. 特に, 会津大学 程子学 講師, 筑波大学 木村成伴 講師, Pairoj Termsinsuwan 氏, Bhed Bista 氏にはお世話になりました.

最後に, これまでつねに励まし続けてくれた父 栄一, 母 ツタ子 に感謝いたします. 両親の支えなしにこの論文は完成しませんでした.

参考文献

[1] Kenneth J. Turner eds. *Using Formal Description Techniques*, p.431, John Wiley & Sons, 1993.

[2] Chris A. Vissers, Richard L. Tenney, and Gregor V. Bochmann. Formal Description Techniques, *Proceedings of the IEEE*, Vol. 71, No. 12, pp. 1356–1364, 1983.

[3] ISO. Information Processing Systems — Open Systems Interconnection — LOTOS — A formal description technique based on the temporal ordering of observational behaviour, IS0/IEC 8807, 1989.

[4] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO Specification Language LOTOS, *Computer Networks and ISDN Systems*, No. 14, pp. 25–59, 1987.

[5] L. Logrippo, M. Faci and M. Haj-Hussein. An introduction to LOTOS: learning by examples, *Computer Networks and ISDN Systems*, No. 23, pp. 325–342, 1992.

[6] ISO. Information Processing Systems — Open Systems Interconnection — ESTELLE — A Formal Description Technique based on the Extended State Transition Model, ISO/IEC 9074, 1989.

[7] Specification and Description Language, CCITT Z. 100, International Consultative Committee on Telegraphy and Telephony, 1988.

[8] Robert M. Keller. Formal Verification of Parallel Programs, *Communications of the ACM*, Vol. 19, No. 7, pp. 371–384, 1976.

[9] R. Milner. *Communication and Concurrency*, Hoare, C. A. R. ed. Prentice Hall International Series in Computer Science, Prentice Hall, 1989.

[10] 稲垣 康善. 抽象データ型の概念と仕様記述法, 情報処理, Vol. 27, No. 2, pp. 120–128, 1986.

[11] 高橋 薫, 神長 裕明, 白鳥 則郎. LOTOS 言語の特質と処理系の現状と動向, 情報処理, Vol. 31, No. 1, pp. 35-46, 1990.

[12] Chris A. Vissers, Giuseppe Scollo, Marten van Sinderen and Ed Brinksma. Specification styles in distributed systems design and verification, *Theoretical Computer Science*, Vol. 89, pp. 179–206, 1991.

[13] Rom Langerak. Decomposition of functionality : a correctness preserving LOTOS transformation, *Proceedings of the Tenth International IFIP WG 6.1 Symposium on Protocol Specification, Testing, and Verification*, Luigi Logrippo, Robert L. Probert and Hasan Ural, pp. 203–218, North-Holland, June, Ottawa, Ontario, Canada, 1990.

[14] ISO. Information Processing Systems — Open Systems Interconnection — Service definition for the Association Control Service Element, IS 8649, 1988.

[15] Hideyoshi Tominaga. Flexible Networks — Introduction —, *Journal of IEICE*, Vol. 77, No. 4, pp. 350–354, 1994.

[16] Norio Shiratori, Kenji Sugawara, Tetsuo Kinoshita and Goutam Chakraborty. Flexible Networks: Basic Concepts and Architecture, *IEICE Transactions on Communication*, Vol. E77-B, No. 11, pp. 1287–1294, 1994.

[17] Norio Shiratori, Kenji Sugawara, Tetsuo Kinoshita and Goutam Chakraborty. Flexible Systems: A Step Towards New Generation Networks, *Proceedings of the 9th International Conference on Information Networking*, pp. 477–482, 1994.

[18] D. R. Kuhn. A Technique for Analyzing the Effects of Changes in Formal Specifications, *The Computer Journal*, Vol. 35, No. 6, pp. 574–578, 1992.

[19] David W. Bustard and Adam C. Winstanley. Making Changes to Formal Specifications: Requirements and an Example, *IEEE Transactions on Software Engineering*, Vol. 20, No. 8, pp. 562–568, 1994.

[20] Saeko Matsuura and Shinichi Honiden. Mapping Specification Change Requirement to Program — Program Synthesis using Specification Change Process —, *Trans. of IPSJ*, Vol. 36, No. 5, pp. 1211–1227, 1995.

[21] Ed Brinksma. Constraint-oriented specification in a constructive formal description technique, *Lecture Notes in Computer Science, 430*, pp. 130–152, 1989,

[22] Kenneth J. Turner. Constraint-oriented style in LOTOS, *Proceedings of the British Computer Society Workshop on Formal Methods in Standards*, Didcot, UK, 1988.

[23] Kassem Saleh and Robert Probert. A Service-based method for the synthesis of communications protocols, *International Journal of Miňi and Microcomputers*, Vol. 12, No. 3, pp. 97–103, 1990.

[24] Ferthat Khendek, Gregor von Bochmann, Christian Kant, New Results on Deriving Protocol Specifications from Service Specifications, *SIGCOM '89 Symposium Communications Architectures & Protocols, Computer Communications Review*, Vol. 19, No. 4, pp. 136–145, 1989.

[25] Ferthat Khendek and Gregor von Bochmann, Incremental Construction Approach for Distributed System Specifications, *Proceedings of the Sixth international conference on Formal Description Techniques*, Richard L. Tenney, Paul D. Amer and M. Umit Uyar, pp. 89–104, October, Boston, Massachusetts, USA, 1993.

[26] Rom Langerak, Event structures for design and transformation in LOTOS, *Proceedings of the Fourth International Conference on Formal Description Techniques*, Ken Parker and Gordon Rose, pp. 271–286, North-Holland, November, Sydney, 1991.

[27] Maria Hultstrøm, Structural Decomposition, *Proceedings of the 14th International IFIP Synposium on Protocol Specification*

Testing and Verification, Son T. Vuong, pp. 193–209, June, Vancouver, B.C., 1994.

[28] Ed Brinksma, Rom Langerak and Peter Broekroelofs, Functionality Decomposition by Compositional Correctness Preserving Transformation, *Computer Aided Verification, CAV93*, 1993.

[29] S. Pavon, M. Hultstrom, J. Quemada, D. de Frutos and Y. Ortega. Inverse Expansion, *Proceedings of FORTE91: Fourth International Conference on: Formal Description Techniques*, pp. 303–318, 1991.

[30] Samir G. Kelekar and George W. Hart. Synthesis of protocols and protocol converters using the Submodule construction approach, *Protocol Specification, Testing, and Verification, XIII*, A. Danthine, G. Leduc and P. Wolper, pp. 307–322, North-Holland, 1993.

[31] Philip Merlin and Gregor v. Bochmann. On the Construction of Submodule Specifications and Communication Protocols, *ACM Transactions on Programming Languages and Systems*, Vol. 5, No. 1, pp. 1–25, 1983.

[32] Peter van Eijk. Tools for LOTOS Specification Style Transformation, *Formal Description Techniques, II*, pp. 43–51, North-Holland, 1990.

[33] Ed Brinksma, Giuseppe Scollo and Chris Steenbergen. LOTOS Specifications, their implemantations and their tests, *Protocol Specification, Testing, and Verification, VI*, B. Sarikaya and G. v. Bochmann, pp. 349–360, North-Holland, 1987.

[34] Kenneth J. Turner. A LOTOS-Based Developmenmt Strategy, *Formal Description Techniques, II*, S. T. Vuong, pp. 117–132, North-Holland, 1990.

[35] A. Udaya Shankar and Simon S. Lam. A Stepwise Refinement Heuristic for Protocol Construction, *ACM Transactions on Programming Languages and Systems*, Vol. 14, No. 3, pp. 417–461, 1992.

[36] Simon S. Lam and A. Udaya Shankar. A Composition Theorem for Layered Systems, *Protocol Specification, Testing, and Verification, XI*, B.Jonsson, J. Parrow and B. Pehrson, pp. 93–108, North-Holland, 1991.

[37] Guy Leduc. A framework based on implementation relations for implemanting LOTOS specifications, *Computer Networks and ISDN Systems*, No. 25, pp. 23–41, 1992.

[38] Rezki Boumezbeur and Luigi Logrippo. Specifying Telophone Systems in LOTOS, *IEEE Communications Magazine*, pp. 38–45, 1993.

[39] Mohammed Faci and Luigi Logrippo. Formal specification of telephone systems in LOTOS: the constraint-oriented style approach, *Computer Networks and ISDN Systems*, No. 21, pp. 53–67, 1991.

[40] Teruo Higashino. Service Specification and Its Protocol Specifications in LOTOS – A Survey for Synthesis and Execution –, *IEICE Transactions on Fundamentals*, Vol. E75-A, No. 3, pp. 330–338, 1992.

[41] Kenneth J. Turner. An Architectural Semantics for LOTOS, *Protocol Specification, Testing, and Verification, VII*, H. Rudin and C. H. West, pp. 15–28, Elsevier Science Publishers B. V. (North-Holland), 1987.

[42] Bhed Bahadur Bista, Zixue Cheng, Atsushi Togashi and Norio Shiratori. A New Approach for Protocol Synthesis Based on LOTOS, *IEICE Transactions on Fundamentals*, Vol. E77-A, No. 10, pp. 1646–1655, 1994.

[43] Dan Paulson and Yair Wand. A Automated Approach to Information Systems Decomposition, *IEEE Transactions on Software Engineering*, Vol. 18, No. 3, pp. 174–189, 1992.

[44] Andrew P. Moore. The Specification and Verified Decomposition of System Requirements Using CSP, *IEEE Transactions on Software Engineering*, Vol. 16, No. 9, pp. 932–948, 1990.

[45] Yair Wand and Ron Weber. An Ontological Model of an Information System, *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, pp. 1282–1292, 1990.

[46] Robert L. Probert and Kassem Saleh. Synthesis of Communication Protocols: Survey and Assesment, *IEEE Transactions on Computers*, Vol. 40, No. 4, pp. 468–476, 1991.

[47] H. Ichikawa, K. Yamanaka and J. Kato. Incremantal Specification in LOTOS, *Protocol Specification, Testing, and Verification, X*, L. Logrippo, R. L. Probert and H. Ural, pp. 183–196, Elsevier Science Publishers B. V. (North-Holland), 1990.

[48] George W. Hart and Samir G. Kelekar. Automated Repair of Complex Systems by Fault Compensation, *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2, pp. 193–205, 1994.

[49] C. V. Ramamoorthy, Siyi Terry Dong and Yutaka Usuda. An Implementation of an Automated Protocol Synthesizer (APS) and its application to the X.21 Protocol, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 9, pp. 886–908, 1985.

[50] Pitro Zafiropulo, Colin H. West, Harry Rudin, D. D. Cowan and Daniel Brand. Towards Analyzing and Synthesizing Protocols, *IEEE Transactions on Communications*, Vol. COM-28, No. 4, pp. 651–661, 1980.

[51] M. Hakan Erdogmus and Robert Johnston. On the Specification and Synthesis of Communicating Processes, *IEEE Transactions on Software Engineering*, Vol. 16, No. 12, pp. 1412–1426, 1990.

[52] Gregor v. Bochmann. Protocol Specification for OSI, *Computer Networks and ISDN Systems*, No. 18, pp. 167–184, 1990.

[53] Simon S. Lam and A. Udaya Shankar. A Theory of Interfaces and Modules I – Composition Theorem, *IEEE Transactions on Software Engineering*, Vol. 20, No. 1, pp. 55–71, 1994.

[54] Gregor v. Bochmann. Finite State Description of Communication Protocols, *Computer Networks*, No. 2, pp. 361–372, 1978.

[55] V. Carchiolo, A. Faro and D. Giordano. Formal description techniques and automated protocol synthesis, *Information and Software Technology*, August, Vol. 34, No. 8, pp. 513–521, 1992.

[56] Joachim Parrow. Submodule Construction as Equation Solving in CCS, *Theoretical Computer Science*, Vol. 68, pp. 175–202, 1989.

[57] M. van Sindern, L. Ferreira Pires and C. A. Vissers. Protocol Design and Implementation Using Formal Methods, *The Computer Journal*, Vol. 35, No. 5, pp. 478–491, 1992.

[58] Hafedh Mili, Fatama Mili and Ali Mili. Reusing Software: Issues and Research Directions, *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, pp. 528–562, 1995.

[59] 岡野浩三. 順序機械型プログラムの階層的設計と分散実行プログラム群への変換, 大阪大学基礎工学部博士学位論文, p. 128, 1995.

著者発表論文

学会論文誌

(1) 郷 健太郎, 白鳥 則郎. 等価性に基づくLOTOS仕様の記述スタイル変換法, 情報処理学会論文誌, Vol. 34, No. 6, pp. 1281–1289, 1993.

(2) 郷 健太郎, 白鳥 則郎. プロセス代数仕様の等価性に基づく分割法の拡張, 電子情報通信学会論文誌 B–I, Vol. J78–B–1, No. 10, pp. 567–573, 1995.

(3) Kentaro Go and Norio Shiratori. A Decomposition of a Formal Specification: a Method and an Example, *IEEE Transactions on Software Engineering* (投稿中).

国際会議予稿集

(1) Kentaro Go and Norio Shiratori. Modularization of a Specification in LOTOS, *Proceedings of the International Conference on Network Protocols*, pp. 55–62, San Francisco, IEEE Computer Society, 1993.

(2) Kentaro Go and Norio Shiratori. A New Decomposition Method for Process Specifications, *Proceedings of the International Conference on Information Networking*, pp. 237–242, Osaka, 1994.

(3) Kentaro Go, Kaoru Takahashi, Hiroaki Kaminaga and Norio Shiratori. Automatic Modification of a Protocol Specification based on Changes of a Service Specification, *International Conference on Parallel and Distributed Systems (ICPADS'96)*, Tokyo, 1996 (accepted).

研究会資料

(1) 白鳥 則郎, 郷 健太郎, 山本 潮. 分散協調型ソフトウェア開発環境とその通信ソフトウェアへの応用, 電子情報通信学会情報ネットワーク研究会資料, No. IN91-119, 1991.

(2) 郷 健太郎, 白鳥 則郎. LOTOS 仕様の系統的な分割アルゴリズム, 電子情報通信学会情報ネットワーク研究会資料, No. IN92-39, 1992.

(3) 郷 健太郎, 白鳥 則郎. 通信ソフトウェアのやわらかい開発法に関する一考察, 電子情報通信学会交換システム研究会資料, Vol. SSE95-63, pp. 55–60, 1995.

(4) 唐橋拓史, 郷 健太郎, Goutam Chakraborty, 菅原研次, 白鳥 則郎. やわらかいネットワークにおけるユーザ要求の獲得に関する一考察, 情報処理学会マルチメディア通信と分散処理研究会資料, Vol. 70-14, pp. 81–86, 1995.

ワークショップ資料

(1) 郷 健太郎. 通信ソフトウェアの工学化 — 等価性に基づく LOTOS 仕様の記述スタイル変換法 —, 第 2 回通信ソフトウェアワークショップ予稿集, 刈谷市, 東海セミナセンタ, 1992.

(2) 郷 健太郎. 段階的詳細化にもとづく通信ソフトウェアの開発, 第1回通信ソフトウェア工学時限研究専門委員会予稿集, 工学院大学, 1993.

(3) 郷 健太郎. 通信ソフトウェアのやわらかい開発法に関する研究 ～ 通信ソフトウェアの新たなチャレンジに向けて ～, 第1回通信ソフトウェア研究会予稿集, 札幌, 1995.

全国大会予稿集

(1) 郷 健太郎, 山本 潮, 白鳥 則郎. 分散協調型通信ソフトウェア開発支援システムの構成 (その 1), 第 44 回 (平成 4 年前期) 全国大会講演論文集 (1), 1992.

(2) 山本 潮, 郷 健太郎, 白鳥 則郎. 分散協調型通信ソフトウェア開発支援システムの構成 (その 2), 第 44 回 (平成 4 年前期) 全国大会講演論文集 (1), 1992.

(3) 郷 健太郎, 白鳥 則郎. LOTOS 仕様の等価性に基づく記述スタイル変換法とその応用, 第 46 回 (平成 5 年前期) 全国大会 講演論文集 (1), 1993.

(4) 小山和也, 郷 健太郎, 白鳥 則郎. LOTOS 仕様の再利用性に基づくプロトコル仕様の系統的生成法, 第 46 回 (平成 5 年前期) 全国大会講演論文集 (1), 1993.

(5) 郷 健太郎, 白鳥 則郎. プロセス代数仕様の分割法の提案, 第 49 回 (平成 6 年後期) 全国大会 講演論文集 (1), 1994 (全国大会奨励賞受賞論文).

付録 A

LOTOS の記述スタイル

A.1 はじめに

LOTOS では, 仕様の記述性・理解性を向上させるために, 仕様の記述スタイル (specification styles) について研究が行われている. 代表的な記述スタイルは次の 4 つである [12].

1. モノリシックスタイル (monolithic style)
2. 制約指向スタイル (constraint-oriented style)
3. 状態指向スタイル (resource-oriented style)
4. 資源指向スタイル (state-oriented style)

以下では, これらの記述スタイルを説明するために, システム question/answer service を各記述スタイルで記述する.

A.2 モノリシックスタイル

特徴 システムを 1 つのブラックボックスとみなし, 外部から観測可能なシステムのアクションを時間順序的に記述する.

記述例

```
process QA_service[Q,A]:noexit:=
  Q?q:question; A!q; A?a:answer; q!a; stop
endproc
```

A.3 制約指向スタイル

特徴 システムのアクションの発生を種々の個別的制約 (アクションの発生に寄与する外部的制約) の集まりとして表現する. すなわち, 1 つのブラックボックスをいくつかのブラックボックスに分解し, その局所的な記述と局所間の大域的な関係を記述する.

制約指向スタイルによる仕様記述は, 論理的詳細化の概念を仕様記述作業に導入することになるため, 段階的な要求仕様の獲得に特に有効である [21, 22]. さらに, この記述スタイルによって記述された仕様は, 実システムに独立なものとなるため, 文献 [1] ではこのスタイルで国際標準となるプロトコルを記述するように奨励している.

記述例

```
process QA_service[Q,A]:noexit:=
  ( QA_local[Q] ||| QA_local[A] )
||
  QA_remote[Q,A]
where

  process QA_local[X]:noexit:=
    X?x:question; X?y:answer; stop
  endproc

  process QA_remote[X,Y]:noexit:=
    X?x:question; Y!x; stop ||| Y?y:answer; X!y; stop
  endproc

endproc
```

A.4 状態指向スタイル

特徴 システムの状態を反映して記述する．システムを１つの資源とみなし，外部から観測可能なアクションとそのアクションの発生により変化する内部の状態変化を記述する．

記述例

```
process QA_service[Q,A]:noexit:=
  choice q:question,a:answer [] QA_service1[Q,A](awaitQ,q,a)
where

  process QA_service1[X,Y](s:state,x:question,y:answer):noexit:=
      [s=awaitQ] -> X?x1:question;
        QA_service1[X,Y](pendingQ,x1,y)
  []  [s=pendingQ] -> Y!x;
        QA_service1[X,Y](awaita,x,y)
  []  [s=awaitA] -> Y?y1:answer;
        QA_service1[X,Y](pendingA,x,y1)
  []  [s=pendingA] -> X?y1;
        QA_service1[X,Y](done,x,y)
  []  [s=done] -> stop
  endproc

endproc
```

A.5 資源指向スタイル

特徴 システムの構成要素 (資源) を記述上に反映させる．具体的に通信システムの場合では，各プロトコルエンティティをそれぞれ１つのプロセスに割当て，通信媒体を１つのプロセスに割当てる．

記述例

```
process QA_service[Q,A]:noexit:=
hide CLQ,CLA in
```

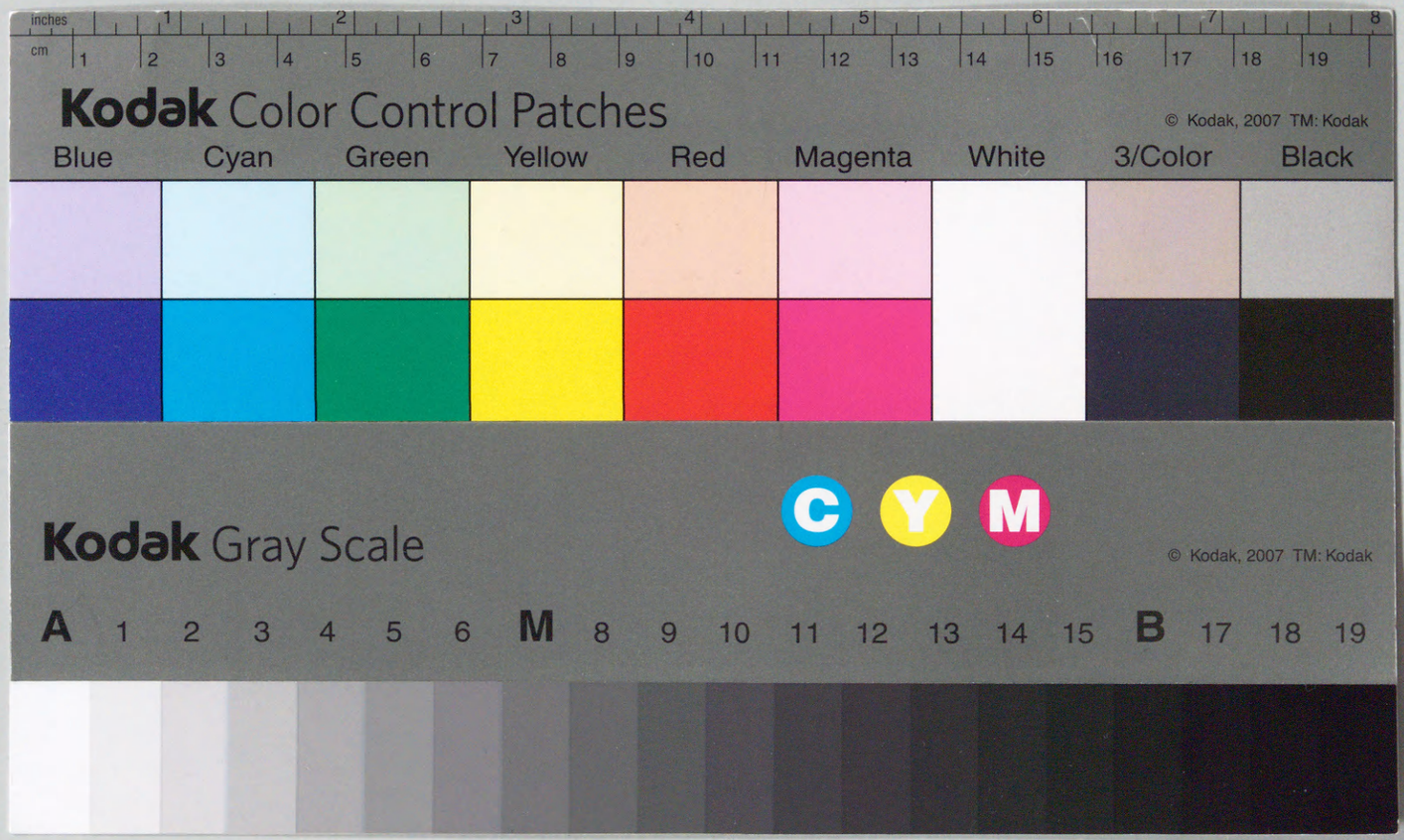
```
  (  (Q_entity[Q,CLQ] ||| A_entity[A,CLA] )
  |[CLQ,CLA]|
     CL_service[CLQ,CLA]
  )
where
  (* process definitions Q_entity,A_entity and CL_service *)
endproc
```

A.6 まとめ

本章では, 付録として仕様の記述スタイルについて述べた. 記述スタイルの観点における本研究の位置付けは次のとおりである.

1) 分割法: モノリシックスタイルの仕様を制約指向スタイルの仕様へ変換する手法に相当する.

2) 変換法: モノリシックスタイルからの資源指向スタイルへの変換法の枠組みの上に構築された手法.


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19